AFOSR-TR-97-0458

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE May 1997	3. REPORT TYPE AND DATES COVERED final : 01 Apr 93 - 31 Mar 97

4. TITLE AND SUBTITLE "Adaptive Reconfiguration Algorithms for Real-Time Distributed Systems"	5. FUNDING NUMBERS contract ~~FY9620-93-1-0029DEF~~ F49620-93-1-0229
6. AUTHOR(S) Aura Ganz, PI, C. M. Krishna and Weibo Gong, co-PIs	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Electrical and Computer Engineering University of Massachusetts/Amherst, MA 01003	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFOSR/NM, Software and Systems 110 Duncan Ave. Suite B115 Bolling AFB, DC 20332-8080	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT DISTRIBUTION UNLIMITED DISTRIBUTION STATEMENT A Approved for public release Distribution Unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

We have developed reconfiguration algorithms that are robust, impose little overhead, and are gracefully degradable and potent enough to extract large performance gains. In addition, we have developed techniques to collect system status information and designed rational approximate techniques to project algorithm performance. We have studied the performance of our reconfiguration algorithm in ring-based and optical networks. Our results promise substantial improvements in distributed real-time system performance without imposing a high overhead.

19971003 058

14. SUBJECT TERMS real-time, configuration, distributed systems, discrete optimization with estimation	15. NUMBER OF PAGES 78
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

DTIC QUALITY INSPECTED 3

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std Z39-18
298-102

Adaptive Reconfiguration Algorithms for Real-Time Distributed Systems

A. Ganz (413) 545-0574

W. B. Gong (413) 545-0384

C. M. Krishna (413) 545-0766

{ganz, gong, krishna}@ecs.umass.edu

Department of Electrical and Computer Engineering

University of Massachusetts

Amherst, MA 01003

Final Report

(Period April 1, 1993 to March 31, 1997)

Contract F49620-93-1-0229DEF.

Contents

Executive Summary

The past few years have seen a trend towards large heterogeneous distributed systems. Such systems are increasingly being used in hard real-time applications such as embedded systems (both defense and civilian) in the control of aircraft, spacecraft, and other processes, for C^3I applications, and in soft real-time applications such as multimedia systems. The performance of real-time systems (probability of meeting critical real-time deadlines) depends on the way in which the various processors are configured (interconnected). The system should be reconfigured for improved performance whenever the computational load changes substantially (e.g., following a change in mission phase), or some hardware failures occur. Reconfiguration has great potential for improving performance, but has largely been overlooked by other researchers.

We have developed reconfiguration algorithms that are robust, impose little overhead, and are gracefully degradable and potent enough to extract large performance gains. In addition, we have developed techniques to collect system status information and designed rational approximant techniques to project algorithm performance. We have studied the performance of our reconfiguration algorithm in ring-based and optical networks. Our results promise substantial improvements in distributed real-time system performance without imposing a high overhead.

We have developed a series of algorithms that deal with discrete optimization problems where the performance index can only be estimated from the real or simulated data. We provided mathematical proof for the convergence of these algorithms. Based on statistical hypothesis testing analysis we provided guidelines for the practical applications of these algorithms in the networking environment. These guidelines include the choice of the comparison schemes in comparing the performance index estimates of two competing configurations. The algorithms and the guidelines developed in the research makes it possible to adaptively optimize the ring-gateway distributed system described in the proposal. We have also obtained a token-based algorithm to control multi-channel networks in order to be able to support the Quality of Service requirements of diverse classes of traffic.

Personnel Supported: Faculty: Aura Ganz, Weibo Gong, and C. M. Krishna. Graduate Students: W. Zhai, A. Yan, Hong Yang, Soracha Nananukul, Kai Yu, Kiran Toutireddy, Rajeev Koodli.

Publications

Journals:

1. Wengang Zhai, P. Kelly and W.B. Gong, "Genetic Algorithms with Noisy Fitness", Mathemetical and Computer Modelling, Volume 23, Number 11/12, pp.131-142, June 1996.

2. W.B. Gong and H. Yang, "Rational Approximation for Some Performance Analysis Problems", Vol. 44, No. 12, pp.1394-1404, IEEE Transactions on Computers, 1995.

3. X.R. Cao, W.B. Gong and Y.Wardi, "Ill-Conditioned Performance Functions of Queueing Systems with Discrete Service Distribu- tions", To appear, IEEE Trans. on Automatic Control, 1995.

4. S. Nananukul and W.B. Gong, "On the MacLauring Series in Light Traffic for a Single Sever Queue", Journal of Applied Pro- bability, March, 1995.

5. C.M. Krishna, A. Ganz, X. Wang, "Window-Based Surveillence Strategies", IEE Proceeding on Computers and Digital Techniques, May 1995.

6. A. Ganz and X. Wang, " Virtual Topology Design of Shared Channel Lightwave Netwroks", to appear in Journal of Fiber and Integrated Optics.

7. A. Ganz and Y. Varoglu, " Time-Wavelength Assignment Algo- rithms: Hardware/Performance Tradeoffs,", to appear in Journal of Fiber and Integrated Optics.

8. A.Ganz, W.B. Gong and X. Wang, "Wavelength Assignment in Multihop Lightwave Networks", IEEE Trans. on Communications, April, 1994.

9. W.B. Gong and H. Yang, "On Global Rational Approximants for Stochastic Discrete Event Systems", Vol. 7, No. 1, International Journal of Discrete Event Dynamic Systems, 1997.

10. W.B. Gong, Y.C. Ho and W. Zhai, "Stochastic Comparison Algorithm for Discrete Optimization with Estimation", Accepted, SIAM Journal on Optimization, 1997.

Refereed Conference Proceedings:

1. W. Zhai and W.B. Gong, "Stochastic Comparison Algorithm for Fan Blade Balancing", Proceedings of the 4th International Conference on Manufacturing, Troy, Oct. 1994.

2. W.B. Gong and H. Yang, "On Global Rational Approximants for Stochastic Discrete Event Systems", Proceedings of the 33rd IEEE Conference on Control and Decision, Lake Buena Vista, Florida, Dec., 1994. (Invited Paper)

3. W.B. Gong and H. Yang, "Approximation of Performance Measures Using Extrapolation Methods", Proceedings of the 33rd IEEE Conference on Control and Decision, Lake Buena Vista, Florida, Dec., 1994. (Invited Paper)

4. H. Yang and W.B. Gong, "A New approach to Calculate Normali- zation Constants in Queueing Networks", Proceedings of the 33rd IEEE Conference on Control and Decision, Lake Buena Vista, Florida, Dec., 1994.

5. Y.M. Jang and A. Ganz " Control Scheme for Broadband Wireless LANs", Seventh In-

ternational Conference on Wireless Communications, Wireless'95, Calgary, Alberta Canada, July 1995.

6. A. Ganz, C. M. Krishna and D. Tang, " Optimal Design of Two- Tier Cellular Radio Systems" Ninth Annual Workshop on Computer Communications, Marathon, FL, October 1994.

PhD Dissertations

Wayne Zhai, Thesis: Discrete Optimization with Estimated Objective Functions: Theory and Applications, May 1995.

Howard Yang, Thesis: Global Rational Approximation for Computer Systems and Communication Networks, Sepetember 1995.

Soracha Nananukul, Thesis: Extrapolations in Stochastic Discrete Event Systems, May 1996.

List of Abbreviations

DOE	Discrete Optimization with Estimation
ISM	Identical Structure Mapping
Mbps	Megabits per second
MPD	Mean Packet Delay
SA	Simulated Annealing
SC	Stochastic Comparison
SR	Stochastic Ruler
RE	Reconfiguration Element
WDM	Wavelength Division Multiplexing

List of Figures

List of Tables

Chapter 1

Introduction

The overall goal of this project is to study reconfiguration of real-time systems in response to system degradation and changing load and priority conditions. Our achievements can be summarized as follows:

- *Development of new optimization algorithms for reconfiguration:* Most optimization algorithms for complex systems involve search. Contemporary algorithms assume the existence of an analytical formula which allows them to quickly and accurately compute the goodness of each point they visit in the search-space. However, distributed systems are too complex for such formulas to exist, and simulation must be used to arrive at a goodness estimate. Such simulations must be short to avoid incurring a high overhead during the search process. Unfortunately, the variance of the estimate from short simulations tends to be large. Thus, the optimization algorithm must not be very sensitive to such variations. We have developed (a detailed description of the optimization techniques for reconfiguration that we have developed is presented in Chapter 2.):
  - A new optimization procedure, which further develops the Stochastic Ruler algorithm, called the *Stochastic Comparison* algorithm. *(Acceleration technique).*
  - An optimization procedure based on genetic algorithms. *(Acceleration technique).*
  - Prediction of optimization quality by means of rational approximants.
- Window-based data techniques to collect system status information (for details see Chapter 3).
- *Implementation of our Algorithms on Real-Time Systems:* We have evaluated and tested the developed techniques in a realistic environment:
  - Token Rings (for details see Chapter 4).

- Optical Networks (for details see Chapter 5).

In each of the following chapters we start by surveying the state-of-the-art developments and proceed by a thorough description of our contributions.

Chapter 2

Discrete Optimization with Estimation Algorithms

2.1 Introduction

Discrete optimization plays an increasingly important role in system design and analysis. Examples include the configuration design of distributed computer systems, routing design, and scheduling problems in communication networks, production systems and transportation systems. The common features of the discrete optimization problems in these systems are (1) the number of feasible alternatives increases exponentially with the system size and (2) the objective function to be optimized does not have an analytical expression. To tackle the first difficulty, one has to relax the goal somewhat. For example, one should only ask for algorithms that provide a good design with high probability (rather than 100% certainty) in realistic time, which is often a reasonable compromise in practice. Simulated annealing (SA) is one such algorithm and it has been successful in solving many practical problems (see, for example, [9]). However, for SA to work well it needs good neighborhood structure and accurate estimates of the objective function. Experiments show that poor choice of neighborhood and rough estimates of the objective function values leads to poor performance [22, 28, 15]. Theoretical analyses of the SA algorithms are extensive. Two types of errors were classified: instantaneous error (the difference between the true and the observed objective function) and accumulated error (the sum of a sequence of the instantaneous errors). Grover presented an early analysis on the effect of the instantaneous error [23]. Durand and White analyzed equilibrium properties for bounded instantaneous error [16]. Gelfand and Mitter showed that under some conditions, slowly decreasing state-independent Gaussian noise does not affect asymptotic convergence [17], [18]. Romeo and Sangiovanni-Vincentelli give conditions on errors such that transition probabilities of a noisy annealing process converge to those of the accurate process as $T \to 0$ [31]. Greening proved a slightly generalized result

[21] by relating the error range and the cooling temperature. All these studies show that SA may work with Monte-Carlo estimates, but it would take extensive computer time to simulate a complex system to get accurate enough estimates. Another algorithm, proposed by Yan and Mukai [34], takes this point into consideration. We call this the Stochastic Ruler (SR) algorithm. In the SR algorithm the estimated objective function is compared with an *a priori* chosen random number called the stochastic ruler. In [34] it is proved that the sequence of feasible alternatives visited by the SR algorithm converges to the optimum in probability under fairly general conditions. In the original SR algorithm, a uniformly distributed random variable over $[a, b]$ is used as the ruler, therefore some *a priori* knowledge regarding the range of the objective function values is needed to determine an appropriate interval size of the ruler. Too big a ruler would reduce the sensitivity of the algorithm and slow down the optimization process. On the other hand, since we don't know *a priori* the location of the "good" candidates, too small a ruler could rule out many such good candidates. In the Stochastic Comparison (SC) algorithm to be analyzed in this chapter we try to directly compare the estimates of the objective function values for two competing alternatives. A preliminary outline of our analysis has been published in [20].

We emphasize that the algorithm presented here is aimed at discrete optimization problems with very big search spaces. Our approach has a root in the ranking and selection procedures well known in statistics (see, for example, [24, 32, 19]). However the typical ranking and selection procedures deal with small size problems and are not concerned with the convergence of the particular iteration procedure we proposed. Similarly, bandit theory deals with similar problems with smaller problem sizes (see, for example, [29], [33] and references therein).

The rest of this chapter is organized as follows. In Section 2.2, we present the discrete optimization with estimation problem and discuss the existing SA and SR algorithms. Section 2.3 proposes the SC algorithm and establish an equivalence result to pave the way for the convergence proof. Section 2.4 analyzes the time-homogeneous Markov chain generated by the SC algorithm when the "testing sequence" is constant. The convergence proof of the SC algorithm via the analysis of the underlying time-inhomogeneous Markov Chain is given in in Section 2.5. Numerical examples are presented in Section 2.6. Finally, we briefly discuss the efficiency of the SC algorithm in Section 2.7.

2.2 The Problem and Existing Algorithms

2.2.1 The Optimization Problem

Since we were originally motivated by a computer system configuration design problem, we will call each element in the search space a *configuration*. Consider a discrete optimization problem where the objective function being optimized is defined over a discrete finite set of configurations, S, i.e.,

$$\min_{i \in S}\{g(i)\}$$

where $g : S \to R$ and $S = \{1, 2, ..., s\}$. The cardinality of set S is denoted by $|S|$ (i.e., s and $|S|$ are synonymous). Furthermore, $g(i)$ can only be evaluated via Monte-Carlo simulation. Usually one uses the sample objective function, denoted by $H(i)$, as the estimate and requires that $g(i) = E[H(i)], \ \forall i \in S$. We also assume that the variance of $H(i)$ is finite, namely, $E[H(i) - E[H(i)]]^2 < \infty, \ \forall i \in S$.

Our goal is to seek a global optimal configuration $i \in S^*$. Here we denote the global optimum set by

$$S^* = \{i \in S \mid \ g(i) \leq g(j), \ \forall j \in S\}. \tag{2.1}$$

Next, we will make some remarks regarding the unbiasedness assumption of the estimate $H(i)$. First, an unbiased estimate is commonly assumed for the theoretical development of stochastic optimization schemes to obtain guidelines for practical applications. This is the case for the algorithm derived in Yan and Mukai [34], which motivated our work here. Second, when the simulated system has a regenerative structure, regenerative simulation can provide unbiased, independent, identically distributed (i.i.d.) samples for the objective function (see, for example, [10], p.95). Although most performance indices are estimated in the form of a ratio estimator (e.g., in a single server queue, the mean waiting time is the ratio of the area under the busy period and the number of departures in that busy period), and thus the regenerative estimator would be biased, we can compare the performance of two configurations in the following manner. Suppose the ratio estimator for configuration a is $\frac{R_a}{Y_a}$ and the one for b is $\frac{R_b}{Y_b}$. Instead of comparing them directly, we should compare $R_a Y_b$ with $R_b Y_a$. Since the regenerative estimator for R_a, R_b, Y_a and Y_b are unbiased, and R_a and Y_b (also R_b and Y_a) are independent, so $E[R_a Y_b] = E[R_a]E[Y_b]$ and $E[R_b Y_a] = E[R_b]E[Y_a]$, and consequently, $E[R_a Y_b] < E[R_b Y_a]$ is equivalent to $\frac{E[R_a]}{E[Y_a]} < \frac{E[R_b]}{E[Y_b]}$. Third, bias reduction techniques, such as jacknifing (see, for example, [10]), can be used to reduce the effect of the bias. Finally, experiments known to us suggest that the guidelines developed under the unbiasedness assumption are indeed useful in practice.

2.2.2 Simulated Annealing and Stochastic Ruler Algorithms

Simulated annealing and stochastic ruler algorithms are the two generic random search algorithms proposed to solve the discrete optimization problem (2.1). To briefly describe these algorithms, we first introduce the following standard assumption and definitions.

Definition 2.2.1 *For each $i \in S$, there exists a subset $N(i)$ of $S \setminus \{i\}$, which is called the* **set of neighbors** *of i.*

Assumption 2.2.1 *For any pair $(i,j) \in S \times S$, j is reachable from i, i.e., there exists a finite sequence $\{n_m\}_{m=0}^{\ell}$ for some ℓ, such that $i_{n_0} = i, i_{n_\ell} = j$, and $i_{n_{m+1}} \in N(i_{n_m})$ for $m = 0,1,2,...,\ell-1$.*

Definition 2.2.2 *A function R: $S \times S \to [0,1]$ is said to be a* **generating probability** *for S and N if*

1. *$R(i,j) > 0 \Longleftrightarrow j \in N(i)$ and*
2. *$\sum_{j \in S} R(i,j) = 1$ for $i,j \in S$.*

In SA, it has been proved ([30]) that a real sequence $\{T_k\}_{k=0}^{\infty}$ satisfying $T_k = \frac{\gamma}{\log(k+k_0+1)}$, $k = 0,1,2,\ldots$ for some positive numbers γ and k_0 will guarantee that the algorithm will converge to a global optimum. T_k is called the *temperature* at the k-th iteration and the sequence $\{T_k\}_{k=0}^{\infty}$ is called the *cooling schedule.* In SR, it has been proved ([34]) that an integer sequence $\{M_k\}_{k=0}^{\infty}$ satisfying $M_k = \lfloor c \log_\sigma(k + k_0 + 1) \rfloor, k = 0,1,2,\ldots$ for some positive numbers c, σ and k_0 will guarantee that the algorithm will converge to a global optimum. We call M_k the k-th *testing number* and $\{M_k\}_{k=0}^{\infty}$ the *testing sequence.*

Let X_k denote the configuration visited by the algorithm at the k-th iteration. Then $\{X_k\}_{k=1}^{\infty}$ is a Markov chain. The fundamental work in SA and SR is to prove the convergence of the probability vector for $\{X_k\}$, $e(k) = [e_1(k) \ldots e_s(k)]$ with $e_i(k) \triangleq \Pr\{X_k = i\}$ for $i = 1, \ldots, s$, to an *optimal probability vector* $e^* = [e_1^* \ldots e_s^*]$, i.e.,

$$e_i^* > 0 \quad \text{for} \quad i \in S^*$$

$$e_i^* = 0 \quad \text{for} \quad i \notin S^*.$$

This was done via application of the weak and strong ergodicity theory of time-inhomogeneous Markov chains ([9], [49], [27], [30]). To guarantee the convergence, the temperature or the testing number has to change as slow as a logarithmic function of the iteration number k ([49], [30], [34]).

We now describe the Markov chain $\{X_k\}$ for SA and SR. In SA, the one-step transition probabilities of the Markov chain $\{X_k\}$ for a given temperature T are

$$P_{ij}(T) = \begin{cases} R(i,j)\min[1, e^{-\{g(j)-g(i)\}/T}] & \text{if } j \in N(i); \\ 1 - \sum_{n \in N(i)} P_{in}(T) & \text{if } j = i; \\ 0 & \text{otherwise.} \end{cases} \tag{2.2}$$

It can be seen that $P_{ij}(T) > 0$, even when $g(j) > g(i)$. In other words, SA visits poor configurations with positive probability in order to jump out of the local minima trap. Moreover, it does so less frequently as the optimization process proceeds so as to simulate the "annealing" process.

In the SR algorithm, the samples of the objective function are compared to an *a priori* chosen random variable. The range of this random variable should cover all "good" values of the objective function samples. Recall that $[a, b]$ is the range, $\Theta(a, b)$ is the random variable (the *stochastic ruler*) that has a uniform distribution over $[a, b]$, and $P(i, a, b)$ is defined as

$$P(i, a, b) = P[H(i) \leq \Theta(a, b)].$$

In SR, the one-step state transition probabilities of the Markov chain $\{X_k\}$ for a given testing number M are

$$P_{ij}(M) = \begin{cases} R(i,j)\{P(j,a,b)\}^M & \text{if } j \in N(i); \\ 1 - \sum_{n \in N(i)} P_{in}(M) & \text{if } j = i; \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

$P_{ij}(M)$ is just the probability that the search goes from configuration i to configuration j when the testing number is M.

2.2.3 Some Remarks

We make some remarks regarding the practical use of SA and SR algorithms in the context of optimizing a testbed system. First, our experiments indicate that SA would not converge for poor objective function estimates. This suggests that SA needs long simulation runs. SR seems to be more robust against the estimation error. This is due to the fact that the Monte-Carlo estimates of order statistics are robust against estimation error. Algorithms based on this observation may significantly reduce the required simulation time [26]. Second, the SA algorithm often visits configurations with poor performance in order to reach a good configuration, surrounded by poor configurations. This is only beneficial when one can choose good neighborhood structure, which is often a difficult task. Third, as we mentioned before, in the SR algorithm, one has to choose a stochastic ruler with appropriate size. Since this could be difficult in practice, we chose the method of directly comparing the estimates of the objective function values for two competing configurations for the *Stochastic Comparison* algorithm. Convergence of the SC algorithm is only guaranteed when any of the configuration, in the whole configuration space, can be reached in one step. Or in other words, we chose to eliminate the neighborhood structure for the sake of convergence. This is not too bad for practical applications, like the computer system configuration design problem mentioned before, since in many such applications a good neighborhood structure is hard to find. Finally, although in the SR algorithm one can also take the whole search space as the neighborhood, we believe that in this case the SC algorithm is clearly more efficient than the SR algorithm, at least when the noise is small. This can be seen by assuming that the noise level is zero. When the performance values are exactly known, SC degenerates to the usual crude random search, while in SR one still compares the performance values with a *random* ruler to decide whether or not to move. The latter scheme is apparently not as efficient as the SC algorithm, because there is no need to introduce the randomness in comparing alternatives, since we are not worrying about "local minima".

One may question the effectiveness of searching a huge space without following any neighborhood structure. The following calculation may help to demonstrate the fact that random search without neighborhood structure could be very effective.

Our algorithm is actually a stochastic version of the crude random search algorithm proposed by Brooks[11]. Brooks's algorithm assumes that the exact value of objective function is available, and thus has a straightforward convergence proof. We calculate the performance of an iterative version of Brooks' algorithm to show that random search without neighborhood structure could be very effective.

Assume that $g(i)$, $\forall i \in S$ is known exactly. In each iteration of the algorithm we simply compare a randomly chosen configuration, which implies $R(j,i) = 1/(|S|-1)$, $\forall j \in S \setminus \{i\}$, with the current configuration. The one having better performance becomes the new "current configuration". When $|S|$, the cardinality of the search space is sufficiently large, the probability of finding one of the top $\alpha\%$ configuration in k iterations is estimated by

$$P_\alpha(k) \approx 1 - \left(1 - \frac{1}{|S|-1}\right)^{0.01\alpha|S|k} \approx 1 - \exp(-0.01\alpha k).$$

The derivation of this estimation is straightforward. Let $\frac{1}{|S|-1}$ be the probability that a top $\alpha\%$ configuration is chosen, then $(1 - \frac{1}{|S|-1})$ would be the probability that this configuration is not chosen, and $(1 - \frac{1}{|S|-1})^{0.01\alpha|S|}$ would be the approximate of the probability that none of the top $\alpha\%$ configuration is chosen. If this happens in k iterations, then the probability would be $(1 - \frac{1}{|S|-1})^{0.01\alpha|S|k}$ and $1 - (1 - \frac{1}{|S|-1})^{0.01\alpha|S|k}$ will give the estimated probability that at least one of the top $\alpha\%$ configurations will be chosen. Once a good configuration is picked, it'll be kept until a better one is found.

Depending on the given problem, α can take different value. As an example, we take $\alpha = 0.5, k = 1000$, then $P_{0.5}(1000) \approx 1 - e^{-5} = 0.99326$ for large $|S|$. This indicates that the crude random search (i.e., without neighborhood structure) could be useful in some applications. This fact is widely used in sampling and is a starting point for the SC algorithm.

We now turn to the detailed description of the SC algorithm.

2.3 The Stochastic Comparison Algorithm

2.3.1 An Alternative Problem

In the Stochastic Comparison algorithm we actually maximize an alternative objective function. We call this function the *sigma-probability* function and denote it by $sp(\cdot)$. For each configuration $i \in S$, we define

$$sp(i) = \textstyle\sum_{j\in S\setminus\{i\}} \Pr\left[H(i) < H(j)\right].$$

The optimum set $\bar{S}^*$ is defined as

$$\bar{S}^* = \{i \in S \mid \; sp(i) \geq sp(j), \;\forall j \in S\}.$$

Under the following assumption (Assumption 2.3.1), this problem is equivalent to the original discrete optimization problem defined by (2.1). In the following, we denote the estimation error $H(i) - g(i)$ by W_i, $\forall i \in S$.

Assumption 2.3.1 *(1) $\{W_i, i \in S\}$ are independent, identically distributed;*

(2) Each $W_i, i \in S$ has a symmetric continuous probability density function and a zero mean.

We first establish the equivalence property.

Theorem 2.3.1 *Under Assumption 2.3.1,*

$$E[H(i)] < E[H(j)] \iff sp(i) > sp(j) \;\forall i \neq j, \; i, j \in S.$$

Proof. Note that we can express $H(i)$ as $H(i) = g(i) + W_i$, where $g(i) = E[H(i)]$, $\forall i \in S$. Let $c = g(i) - g(j)$.

By Assumption 2.3.1, we have that $W_j - W_i, W_k - W_i$ and $W_k - W_j$, $\forall k \in S \setminus \{i, j\}$ are identically distributed random variables with symmetric continuous density function. Let ξ be a random variable with the same density. Then we have

$$\begin{aligned} sp(i) &= \Pr[H(i) < H(j)] + \sum_{k\in S\setminus\{i,j\}} \Pr[H(i) < H(k)] \\ &= \Pr[W_j - W_i > g(i) - g(j)] + \sum_{k\in S\setminus\{i,j\}} \Pr[W_k - W_i > g(i) - g(k)] \\ &= \Pr[\xi > c] + \sum_{k\in S\setminus\{i,j\}} \Pr[\xi > c + g(j) - g(k)] \end{aligned}$$

and

$$\begin{aligned} sp(j) &= \Pr[H(j) < H(i)] + \sum_{k\in S\setminus\{i,j\}} \Pr[H(j) < H(k)] \\ &= \Pr[W_j - W_i < g(i) - g(j)] + \sum_{k\in S\setminus\{i,j\}} \Pr[W_k - W_j > g(j) - g(k)] \\ &= \Pr[\xi < c] + \sum_{k\in S\setminus\{i,j\}} \Pr[\xi > g(j) - g(k)]. \end{aligned}$$

Since $\Pr[\xi < c] = 1 - \Pr[\xi \geq c] = 1 - \Pr[\xi > c]$, we have

$$sp(i) - sp(j) = 2\Pr[\xi > c] - 1 + \sum_{k\in S\setminus\{i,j\}} \{\Pr[\xi > c + g(j) - g(k)] - \Pr[\xi > g(j) - g(k)]\}.$$

Note that the right hand side of the above equation is a monotonic decreasing function of c. Note also that $sp(i) - sp(j) = 0$ when $c = 0$. Therefore, we have $sp(i) > sp(j) \Longleftrightarrow E[H(i)] < E[H(j)]$. Q.E.D.

In the following, we present a Corollary of Theorem 2.3.1 and a Lemma for later use.

Corollary 2.3.1 *Given $H(i)$, $H(j)$ $\forall i \neq j$, $i,j \in S$, we have under Assumption 2.3.1:*

$$E[H(i)] < E[H(j)] \Longleftrightarrow \Pr[H(i) < H(j)] > \Pr[H(j) < H(i)].$$

Proof. Let $S = \{i,j\}$ and apply Theorem 2.3.1. Q.E.D.

Lemma 2.3.1 *Given $H(i)$, $H(j)$, $H(k)$ $\forall i \neq j, j \neq k, k \neq i$, $i,j,k \in S$, the following two conditions are equivalent under Assumption 2.3.1:*

1. *$E[H(i)] < E[H(j)] < E[H(k)]$;*
2. *$\Pr[H(i) < H(k)] > \Pr[H(j) < H(k)]$ and $\Pr[H(i) < H(k)] > \Pr[H(i) < H(j)]$.*

Proof. Using the same notations as in the proof of Theorem 2.3.1, we have

$$\begin{aligned} \Pr[H(i) < H(k)] &= \Pr[W_i - W_k < g(k) - g(i)] = \Pr[\xi < g(k) - g(i)] \\ \Pr[H(j) < H(k)] &= \Pr[W_j - W_k < g(k) - g(j)] = \Pr[\xi < g(k) - g(j)] \\ \Pr[H(i) < H(j)] &= \Pr[W_i - W_j < g(j) - g(i)] = \Pr[\xi < g(j) - g(i)]. \end{aligned}$$

The results of the lemma follow immediately. Q.E.D.

In the SC algorithm, we request that any of the configurations can be reached in one step with positive probability. Hence we have,

Assumption 2.3.2 $R(i,j) > 0$, $\forall i,j \in S$

2.3.2 Implementation of the Stochastic Comparison Algorithm

We now describe the implementation of the SC algorithm.

Let S be the configuration space, k be the iteration number, X_k be the configuration accepted at iteration k. Recall that M_k represents the number of samples required in the k-th iteration and samples of $H(i)$ are denoted by $H_\ell(i)$ for $\ell = 1, \ldots, M_k$.

The Stochastic Comparison Algorithm

Data : R, $\{M_k\}$, $i_0 \in S$.

Step 0: Set $X_0 = i_0$ and $k = 0$;

Step 1: Given $X_k = i$, choose a candidate Z_k from $S \setminus \{i\}$ with probability distribution

$$P[Z_k = j | X_k = i] = R(i,j), \qquad j \in S \setminus \{i\}$$

Step 2: Given $Z_k = j$, set

$$X_{k+1} = \begin{cases} Z_k, & \text{if } H_\ell(j) < H_\ell(i), \ \forall \ell = 1, \ldots, M_k \\ X_k, & \text{otherwise} \end{cases} ;$$

Step 3: Set $k = k+1$ and go to Step 1.

The implementation of Step 2 can be described as follows. We draw samples from both $H(i)$ and $H(j)$. If $H(j) \geq H(i)$, then set $X_{k+1} = X_k$ and go to step 3; if $H(j) < H(i)$, then draw another pair of samples and compare them again. If for all the M_k samples we have $H(j) < H(i)$, then set $X_{k+1} = Z_k$ and go to Step 3. Due to the i.i.d. assumption of $\{H_\ell(i), \ell = 1, \cdots, M_k;\ i \in S\}$, the state transition probability from i to j is

$$R(i,j) \Pr[H_1(j) < H_1(i), \cdots, H_{M_k}(j) < H_{M_k}(i)] = R(i,j)\{\Pr[H(j) < H(i)]\}^{M_k}$$

2.3.3 Sketch of the Convergence Proof

If the objective function is known exactly, then it is trivial to prove that the SC algorithm converges to an optimal configuration. The convergence analysis becomes involved when the objective function has to be *estimated.*

From the implementation of the algorithm, we see that those configurations visited by the SC algorithm in successive steps form a time-inhomogeneous Markov chain $\{X_k\}$. Our convergence analysis is based on results of time-inhomogeneous Markov chain theory in [27]

The outline of our analysis is as follows.

1. Set $M_k = M$ and study the corresponding Markov chain at its steady state;
2. Let M go to infinity and show that
   (a) the $\pi(M)$ converges to an optimal probability vector; and that
   (b) the $\pi(M)$ has monotone property with respect to M for large enough M;
3. Show that the Markov chain with $M_k = M$ is weak ergodic by calculating the coefficient of ergodicity. We will give more intuitive explanation in a later section;
4. Show that the Markov chain with $M_k = M$ is strong ergodic based on a lemma and the weak ergodicity;
5. Show the convergence of the Markov chain $\{X_k\}$ based on the strong ergodicity and a theorem on time-inhomogeneous Markov chain.

Essentially the convergence proof for the SC algorithm follows the same lines as those for SA and SR algorithms, however, the main component of the proof (the second step), or the convergence of the $\pi(M)$ vector to an optimal probability vector in the SC algorithm, is very different from those for SA and SR algorithms. In the next section, Section 2.4, we discuss the quasi-stationary probabilities of the Markov Chains that describe the SC algorithm. The convergence of the SC algorithm is proved in Section 2.5. We point out that the Markov chain describing the SC algorithm belongs to the class studied by Connors and Kumar in [14].

2.4 Markov Chain under Constant Testing Number

2.4.1 Markov Chain Equations

The one-step state transition probabilities of a Markov chain $\{X_k\}$ for a given testing number M are

$$P_{ij}(M) = \begin{cases} R(i,j)\{\Pr[H(j) < H(i)]\}^M & \text{if } j \neq i; \\ 1 - \sum_{n=1,n\neq i}^{s} P_{in}(M) & \text{if } j = i. \end{cases} \tag{2.4}$$

We denote $r_{ij} = R(i,j)$, $p_{ij} = \Pr[H(j) < H(i)]$ and $t_{ij} = r_{ij}p_{ij}^M$ $(i \neq j)$ to simplify the notations. The one-step state transition probability matrix $P(M)$ can be written as

$$P(M) = \begin{bmatrix} 1 - \sum_{n=2}^{s} t_{1n} & t_{12} & t_{13} & \cdots & t_{1s} \\ t_{21} & 1 - \sum_{n=1,n\neq 2}^{s} t_{2n} & t_{23} & \cdots & t_{2s} \\ t_{31} & t_{32} & 1 - \sum_{n=1,n\neq 3}^{s} t_{3n} & \cdots & t_{3s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_{s1} & t_{s2} & t_{s3} & \cdots & 1 - \sum_{n=1}^{s-1} t_{sn} \end{bmatrix}.$$

Let

$$A = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ t_{12} & -\sum_{n=1,n\neq 2}^{s} t_{2n} & t_{32} & \cdots & t_{s2} \\ t_{13} & t_{23} & -\sum_{n=1,n\neq 3}^{s} t_{3n} & \cdots & t_{s3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_{1s} & t_{2s} & t_{3s} & \cdots & -\sum_{n=1}^{s-1} t_{sn} \end{bmatrix}$$

$$= \begin{bmatrix} a_1 & a_2 & a_3 & \cdots & a_s \end{bmatrix}$$

and

$$b = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \end{bmatrix}^T.$$

We will show that there exists a vector $\pi(M) = [\pi_1(M), \pi_2(M), \cdots, \pi_s(M)]$ that satisfies

$$A\pi^T(M) = b.$$

We denote matrix

$$B_m = \begin{bmatrix} a_1 & \cdots & a_{m-1} & b & a_{m+1} & \cdots & a_s \end{bmatrix},$$

that is, $\{B_m\}_{m=1}^{s}$ is obtained via replacing m-th column of matrix A by vector b.

Let $|X|$ represent the determinant of matrix X. It can be verified that $|A|$ and $\{|B_m|\}_{m=1}^{s}$ are related by

$$|A| = |B_1| + |B_2| + \cdots + |B_s|. \tag{2.5}$$

With Assumption 2.3.2, any configuration can be picked in the next step with positive probability, therefore the Markov Chain, $\{X_k\}$, generated by the SC algorithm is irreducible and positive recurrent. Furthermore, its stationary distribution exists and is unique. From Lemma 2.4.2 and equation (2.5), we have that $|A| \neq 0$. By Cramér's Rule,

$$\pi_1(M) = \frac{|B_1|}{|A|},\ \pi_2(M) = \frac{|B_2|}{|A|}, \cdots,\ \pi_s(M) = \frac{|B_s|}{|A|}.$$

Since $\pi(M)$ satisfies $\pi(M) = \pi(M)P(M)$ and $\sum_{i\in S}\pi_i(M) = 1$, they must be the unique stationary probability distribution defined by the state transition probability matrix $[P_{ij}(M)]$. We also call $\pi(M)$ the *quasi-stationary probability distribution* of the time-inhomogeneous Markov chain $\{X_k\}$ ([30]).

We now expand each $|B_i|$, $\forall i \in S$ along its i-th column. The resulting expansions have very similar form, as stated in the following Lemma.

Lemma 2.4.1 *For $i \in S$, $|B_i|$, each has an equivalent form as follows.*

1. For $i = 1$,

$$|B_1| = \begin{vmatrix} -\sum_{n=1,n\neq 2}^{s} t_{2n} & t_{32} & \cdots & t_{s2} \\ t_{23} & -\sum_{n=1,n\neq 3}^{s} t_{3n} & \cdots & t_{s3} \\ \vdots & \vdots & \vdots & \vdots \\ t_{2s} & t_{3s} & \cdots & -\sum_{n=1}^{s-1} t_{sn} \end{vmatrix}.$$

2. For $1 < i < s$,

$$|B_i| = \begin{vmatrix} -\sum_{n=2}^{s} t_{1n} & \cdots & t_{(i-1)1} & t_{(i+1)1} & \cdots & t_{s1} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ t_{1(i-1)} & \cdots & -\sum_{n=1,n\neq i-1}^{s} t_{(i-1)n} & t_{(i+1)(i-1)} & \cdots & t_{s(i-1)} \\ t_{1(i+1)} & \cdots & t_{(i-1)(i+1)} & -\sum_{n=1,n\neq i+1}^{s} t_{(i+1)n} & \cdots & t_{s(i+1)} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ t_{1s} & \cdots & t_{(i-1)s} & t_{(i+1)s} & \cdots & -\sum_{n=1}^{s-1} t_{sn} \end{vmatrix}.$$

3. For $i = s$,

$$|B_s| = \begin{vmatrix} -\sum_{n=2}^{s} t_{1n} & t_{21} & \cdots & t_{(s-1)1} \\ t_{12} & -\sum_{n=1,n\neq 2}^{s} t_{2n} & \cdots & t_{(s-1)2} \\ \vdots & \vdots & \ddots & \vdots \\ t_{1(s-1)} & t_{2(s-1)} & \cdots & -\sum_{n=1,n\neq s-1}^{s} t_{(s-1)n} \end{vmatrix}.$$

Proof. Expand each $|B_i|$, $\forall i \in S$ along its i-th column. The resulting determinant is also denoted by $|B_i|$. If $i = 1$, then *1.* follows immediately. If $1 < i \leq s$, then for every $\ell = 1, \ldots, i-1$, multiply the ℓ-th row by -1 and subtract the rest of rows from the ℓ-th row. Note that the expansion introduces $i+1$ times sign changes, and the multiplication introduces $i-1$ times sign changes, therefore there is no sign change. We obtain *2.* and *3.* after performing the aforementioned multiplications and subtractions.

Q.E.D.

We further expand each $|B_i|$, $\forall i \in S$ into a summation form, such that each term in the summation is a product of $(s-1)$ elements chosen from $\{t_{ij} \mid i,j \in S\}$ according to the expansion theorem. The subscripts of t_{ij}, the i and the j, represent two configurations. To refer to a particular t_{ij}, we have to clearly indicate its subscripts i, j, especially when there are many t_{ij}'s involved. An element of a permutation set is a collection of such subscripts that were arranged in a specific order. To name a given permutation, we assign it a unique index. An index set refers to all the permutations it represents. For example, if $S = \{1,2,3\}$ and $S^* = \{1\}$, then

$PS_1 = \{\{2,3\},\{3,2\}\}$

$PS_1' = \{\{2,3\},\{3,2\}\} \setminus \{2,3\} = \{\{3,2\}\}$

$IPS_1 = \{A_1, A_2 | A_1 \textit{ represents } \{2,3\}, A_2 \textit{ represents } \{3,2\}\}$

$IPS_1' = \{A_2 | A_2 \textit{ represents } \{3,2\}\}$

$QS_1 = \{\{1,1\},\{1,2\},\{3,1\},\{3,2\}\}$

$QS_1' = QS_1 \setminus \{1,1\} = \{\{1,2\},\{3,1\},\{3,2\}\}$

$IQS_1 = \{B_1, B_2, B_3, B_4 | B_1 \textit{ represents } \{1,1\}, B_2 \textit{ represents } \{1,2\},$
$B_3 \textit{ represents } \{3,1\}, B_4 \textit{ represents } \{3,2\}\}$

$IQS_1' = \{B_2, B_3, B_4 | B_2 \textit{ represents } \{1,2\}, B_3 \textit{ represents } \{3,1\}, B_4 \textit{ represents } \{3,2\}\}$

$OS_1 = \{\{1,1\}\}$

$IOS_1 = \{C_1 | C_1 \textit{ represents } \{1,1\}\}$

NS_1, INS_1 can be expressed similarly and are omitted here.

Definition 2.4.1 *For each $i = 1, \ldots, s$, we define a permutation set and its index set:*

1. *Permutation set*

   $PS_i \triangleq \{\{k_j\}_{j=1,j\neq i}^{s} \mid \{k_j\}_{j=1,j\neq i}^{s} \textit{ is a permutation of } s-1 \textit{ integers in } \{1,\ldots,s\} \setminus \{i\}\}$

   $PS_i' \triangleq PS_i \setminus \{\{k_j\}_{j=1,j\neq i}^{s} | k_j = j\}.$

   $IPS_i \triangleq \{\ell \mid \ell \textit{ is an index for each } \{k_j\}_{j=1,j\neq i}^{s} \in PS_i\}$

   $IPS_i' \triangleq \{\ell \mid \ell \textit{ is an index for each } \{k_j\}_{j=1,j\neq i}^{s} \in PS_i'\}.$

2. *Combination set*

   $QS_i \triangleq \{\{k_j\}_{j=1,j\neq i}^{s} \mid k_j \in \{1,\ldots,s\} \setminus \{j\}, j = 1,\ldots,s, j \neq i\}\}$

   $QS_i' \triangleq QS_i \setminus \{\{k_j\}_{j=1,j\neq i}^{s} | k_j = i\}.$

   $IQS_i \triangleq \{\ell \mid \ell \textit{ is an index for each } \{k_j\}_{j=1,j\neq i}^{s} \in QS_i\}$

   $IQS_i' \triangleq \{\ell \mid \ell \textit{ is an index for each } \{k_j\}_{j=1,j\neq i}^{s} \in QS_i'\}.$

3. *Optimum combination set*

   $OS_i \triangleq \{\{k_j\}_{j=1,j\neq i}^{s} \in QS_i \mid k_j \in S^*,\ \textit{for each } j = 1,\dots,s, j \neq i\}$

   $IOS_i \triangleq \{\ell \mid \ell \textit{ is an index for each } \{k_j\}_{j=1,j\neq i}^{s} \in OS_i\}.$

4. *Non-optimum combination set*

   $NS_i \triangleq QS_i \setminus OS_i$

   $INS_i \triangleq IQS_i \setminus IOS_i.$

Lemma 2.4.2 *Let* $\mathcal{R}_i = \prod_{j=1,j\neq i}^{s} r_{ji}$ *and* $\mathcal{P}_i = \prod_{j=1,j\neq i}^{s} p_{ji}$, $\forall i \in S$. *Let also* $\mathcal{R}_i^{(\ell)} = \prod_{j=1,j\neq i}^{s} r_{jk_j}$ *and* $\mathcal{P}_i^{(\ell)} = \prod_{j=1,j\neq i}^{s} p_{jk_j}$, $\forall i \in S$, *where* $\{k_j\}_{j=1,j\neq i}^{s} \in QS_i'$ *and* $\ell \in IQS_i'$.

Then the expansion of $|B_i|$, $\forall i \in S$ *has the following properties:*

1. $|B_i| = (-1)^{(s-1)}\Big[\mathcal{R}_i\mathcal{P}_i + \sum_{\ell\in IQS_i'} C_i^{(\ell)}\mathcal{R}_i^{(\ell)}\mathcal{P}_i^{(\ell)}\Big]$, *where* $C_i^{(\ell)}$ *is the number of times that* $\mathcal{R}_i^{(\ell)}\mathcal{P}_i^{(\ell)}$ *appears in the summation;*
2. $(-1)^{(s-1)}|B_i| > 0$;
3. $\forall i \in S^*$, $\mathcal{P}_i = \mathcal{P}_i^{(\ell)}$, *if* $\ell \in IOS_i$;
4. $\forall i \in S^*$, $\mathcal{P}_i > \mathcal{P}_i^{(\ell)}$, *if* $\ell \in INS_i$;
5. $\forall i \in S^*$, $\mathcal{P}_i > \mathcal{P}_n$, *if* $n \in S \setminus S^*$;
6. $\forall i \in S^*$, $\mathcal{P}_i > \mathcal{P}_n^{(\ell)}$, *if* $n \in S \setminus S^*$.

Proof. Let $t_{ii} = -\sum_{n=1,n\neq i}^{s} t_{in}$ and $\mathcal{U}_i^{(\ell)} = \prod_{j=1,j\neq i}^{s} t_{jk_j}$, where $\{k_j\}_{j=1,j\neq i}^{s} \in PS_i'$ and $\ell \in IPS_i'$. Let also $ncs(\ell)$ be the number of sign changes for $\mathcal{U}_i^{(\ell)}$.

When applying the Laplace expansion theorem of determinant to $|B_i|$, we have

$$|B_i| = (-1)^{s-1}\Big[\prod_{j=1,j\neq i}^{s}\Big(\sum_{n=1,n\neq j}^{s} t_{jn}\Big) + \sum_{\ell\in IPS_i'} (-1)^{s-1+nsc(\ell)}\cdot\mathcal{U}_i^{(\ell)}\Big]$$

If we further expand $|B_i|$ into summation of $\mathcal{R}_i^{(\ell)}\mathcal{P}_i^{(\ell)}$'s, the index set QS_i' will be used. We can find only one $\mathcal{R}_i\mathcal{P}_i$ term from $\prod_{j=1,j\neq i}^{s}\left(\sum_{n=1,n\neq j}^{s} t_{jn}\right)$, and none from the summands of $\sum_{\ell\in IPS_i'}(-1)^{s-1+nsc(\ell)}\cdot\mathcal{U}_i^{(\ell)}$, since the second index of all t_{ij} in $\mathcal{R}_i^{(\ell)}\mathcal{P}_i^{(\ell)}$ can not be i. Note also that all negative $(-1)^{s-1+nsc(\ell)}\mathcal{R}_i^{(\ell)}\mathcal{P}_i^{(\ell)}$'s for $\ell \in IQS_i'$ (if any) in $\sum_{\ell\in IPS_i'}(-1)^{s-1+nsc(\ell)}\mathcal{U}_i^{(\ell)}$ were canceled, because a corresponding term can be found in $\prod_{j=1,j\neq i}^{s}\left(\sum_{n=1,n\neq j}^{s} t_{jn}\right)$. Therefore, we proved *1.* and *2.*.

In the rest of the proof, we will need the results of Corollary 2.3.1, Theorem 2.3.1 and Lemma 2.3.1. For all $i \in S^*$ and $\{k_j\}_{j=1,j\neq i}^{s} \in QS_i'$ with its index $\ell \in IQS_i'$, we have

$$\mathcal{P}_i = \prod_{j=1,j\neq i}^{s} p_{ji} = \prod_{j=1,j\neq i}^{s} \Pr[H(i) < H(j)]$$

$$\mathcal{P}_i^{\ell} = \prod_{j=1,j\neq i}^{s} p_{jk_j} = \prod_{j=1,j\neq i}^{s} \Pr[H(k_j) < H(j)].$$

If $\{k_j\}_{j=1,j\neq i}^{s} \in OS_i'$, then $E[H(i)] = E[H(k_j)]$, $\forall k_j$, which implies $\Pr[H(i) < H(j)] = \Pr[H(k_j) < H(j)]$. Therefore, *3.* is true.

If $\{k_j\}_{j=1,j\neq i}^{s} \in NS_i'$, then $\exists k_{j'}$, such that $E[H(i)] < E[H(k_j)]$, which implies $\Pr[H(i) < H(j)] > \Pr[H(k_{j'}) < H(j)]$. Therefore, *4.* is true.

For all $n \in S \setminus S^*$, we have

$$\mathcal{P}_i = \Pr[H(i) < H(n)] \prod_{j=1,j\neq i,n}^{s} \Pr[H(i) < H(j)]$$

and

$$\mathcal{P}_n = \Pr[H(n) < H(i)] \prod_{j=1,j\neq i,n}^{s} \Pr[H(n) < H(j)].$$

Since $\Pr[H(i) < H(n)] > \frac{1}{2} > \Pr[H(n) < H(i)]$ and $\Pr[H(i) < H(j)] > \Pr[H(n) < H(j)]$, $\forall j \neq i, n$. Therefore, *5.* is true.

Furthermore, we have

$$\mathcal{P}_n^{(\ell)} = \Pr[H(k_i) < H(i)] \prod_{j=1,j\neq i,n}^{s} \Pr[H(k_j) < H(j)].$$

Since $\Pr[H(i) < H(n)] > \frac{1}{2} \geq \Pr[H(k_i) < H(i)]$ and $\Pr[H(i) < H(j)] \geq \Pr[H(k_j) < H(j)]$, $\forall j \neq i, n$. Therefore, *6.* is true.

Q.E.D.

2.4.2 Convergence of $\pi(M)$ to an Optimal Probability Vector

Let $M \to \infty$ and note that, $\mathcal{P}_i^M$ and $[\mathcal{P}_i^{(\ell)}]^M$, $\forall i \in S^*$ and $\forall \ell \in IOS_i$, will dominate all other $[\mathcal{P}_i^{(\ell)}]^M$, $\forall \ell \in INS_i$ and $\mathcal{P}_j^M$, $[\mathcal{P}_j^{(\ell)}]^M$, $\forall j \in S \setminus S^*$ and $\forall \ell \in IQS_j$. Therefore we have

$$\lim_{M\to\infty} \pi_i(M) = \begin{cases} e_i^* > 0 & \text{if } i \in S^* \\ 0 & \text{if } i \in S \setminus S^* \end{cases}$$

Hence as $M \to \infty$, the quasi-stationary probability vector converges to an optimal probability vector.

2.4.3 Monotone Property of the Quasi-stationary Probabilities

From the solution form of $\pi_1(M)$, $\pi_2(M), \cdots, \pi_s(M)$, we see that $\exists M^* < \infty$, such that for $M_k > M^*$ the quasi-stationary probabilities have monotone property, namely,

$$\pi_i(M_{k+1}) > \pi_i(M_k) \quad \text{for} \quad i \in S^*,$$

$$\pi_i(M_{k+1}) < \pi_i(M_k) \quad \text{for} \quad i \in S \setminus S^*.$$

2.5 Convergence of the SC Algorithm

The convergence proof of the SC algorithm is based on theorems in [27] about weak and strong ergodicity of time-inhomogeneous Markov chains.

We use $P_1, P_2, \ldots,$ to represent the one-step state transition probability matrix of a time-inhomogeneous Markov chain, $\{Y_k\}$, with starting probability vector $\mathbf{f}^{(0)}$. Define

$$\mathbf{f}^{(k)} = \mathbf{f}^{(0)} P_1 \cdot P_2 \cdot \ldots \cdot P_k \text{ and } \mathbf{f}^{(m,k)} = \mathbf{f}^{(0)} P_{m+1} \cdot P_{m+2} \cdot \ldots \cdot P_{m+k}.$$

We are interested in the limiting behavior of $\mathbf{f}^{(k)}$, or $\mathbf{f}^{(m,k)}$ for any integer $m < k$, as $k \to \infty$. If $\{\mathbf{f}^{(k)}\}$ converges to the same fixed probability vector, $\mathbf{q}$, no matter what starting vector $\mathbf{f}^{(0)}$ is used, then we say that the Markov chain is strongly ergodic. Such a behavior is often referred to as *convergence and loss of memory.* If, however, for any starting probability vectors $\mathbf{f}^{(0)}$ and $\mathbf{g}^{(0)}$, $\mathbf{f}^{(k)}$ and $\mathbf{g}^{(k)}$, are "close" for sufficiently large k, but $\mathbf{f}^{(k)}$ and $\mathbf{f}^{(k+1)}$ need not to be very "close" for large k, then we say that the Markov chain is weakly ergodic. In this case, the chain has the property of *loss of memory without convergence.*

To give rigorous definitions of weak and strong ergodicity, we first introduce a norm operator $\|\cdot\|$. If $\mathbf{f} = (f_1, f_2, \ldots)$ is a vector, define the norm of $\mathbf{f}$ by

$$\|\mathbf{f}\| = \sum_{i=1}^{\infty} |f_i|.$$

A time-inhomogeneous Markov chain $\{Y_k\}$ is called *weakly ergodic* if $\forall m$,

$$\lim_{k\to\infty} \sup_{\mathbf{f}^{(0)},\mathbf{g}^{(0)}} \|\mathbf{f}^{(m,k)} - \mathbf{g}^{(m,k)}\| = 0,$$

where $\mathbf{f}^{(0)}$ and $\mathbf{g}^{(0)}$ are starting probability vectors.

A time-inhomogeneous Markov chain $\{Y_k\}$ is called *strongly ergodic*, if there exists a probability vector $\mathbf{q}$ such that $\forall m$,

$$\lim_{k\to\infty} \sup_{\mathbf{f}^{(0)}} \|\mathbf{f}^{(m,k)} - \mathbf{q}\| = 0.$$

where $\mathbf{f}^{(0)}$ is a starting probability vector.

2.5.1 Weak Ergodicity

We introduce the following notations for the smallest nonzero $R(i,j)$ and the smallest $\Pr[H(j) < H(i)]$:

$$\rho = \min_{i\in S} \min_{j\in S\setminus\{i\}} R(i,j)$$

$$\mu = \min_{i\in S} \min_{j\in S\setminus\{i\}} \Pr[H(j) < H(i)].$$

From Assumption 2.3.2, we have $\rho > 0$ and from Assumption 2.3.1 we have $0 < \mu < 1$ ($\Pr[H(j) < H(i)] = \Pr[W_i - W_j < g(i) - g(j)] \in (0,1)$). Choose $\sigma \geq \frac{1}{\mu}$, $0 < c < 1$, k_0, such that $1 \leq c\log_\sigma k_0$ and $M_k = \lfloor c\log_\sigma(k + k_0)\rfloor$ for $k = 0, 1, 2, \cdots$, where $\lfloor \xi \rfloor$ denotes the greatest integer smaller or equal to ξ.

Then we have

$$P_{ij}(M_k) = R(i,j)\Pr[H(j) < H(i)]^{M_k} \geq \rho(\frac{1}{\sigma})^{M_k}.$$

Define $P(k+1,k) = P(M_k)\cdot P(M_{k+1})$, the coefficient of ergodicity, ([27]) is

$$\alpha[P(k+1,k)] \triangleq \min_{i,j} \sum_{\ell\in S} \min_{\ell}[P_{i\ell}, P_{j\ell}] \geq \min_{i,j} \min[P_{i\ell'}, P_{j\ell'}] \geq \rho(\frac{1}{\sigma})^{M_k}.$$

It can be seen that

$$\sum_{k=k^*}^{\infty} \alpha(P(k+1,k)) \geq \sum_{k=k^*}^{\infty} \rho(\frac{1}{\sigma})^{M_k} \geq \sum_{k=k^*}^{\infty} \rho(\frac{1}{\sigma})^{c\log_\sigma(k+k_0)} = \sum_{k=k^*}^{\infty} \rho\frac{1}{(k+k_0)^c} \to \infty,$$

if $c \leq 1$. Hence, the Markov chain generated by SC algorithm is weakly ergodic by Theorem V.3.2 of [27].

2.5.2 Strong Ergodicity

With the monotone property of the quasi-stationary probabilities we have,

Lemma 2.5.1 *The probability vector, $\pi(M)$, defined in $\pi(M)P(M) = \pi(M)$ satisfies,*

$$\sum_{k=0}^{\infty} \|\pi_i(M_{k+1}) - \pi_i(M_k)\| < \infty.$$

Proof. It follows from the monotone property of π_i that there exists an integer k^* such that, for any $k > k^*$,

$$\pi_i(M_{k+1}) \geq \pi_i(M_k) \qquad \forall i \in S^*$$
$$\pi_i(M_{k+1}) \leq \pi_i(M_k) \qquad \forall i \in S \setminus S^*.$$

Hence, for any $k \geq k^*$,

$$\|\pi_i(M_{k+1}) - \pi_i(M_k)\| = \sum_{i\in S^*}[\pi_i(M_{k+1}) - \pi_i(M_k)] - \sum_{i\in S\setminus S^*}[\pi_i(M_{k+1}) - \pi_i(M_k)].$$

Note that from $\sum_{i\in S^*}\pi_i(M_k) + \sum_{i\in S\setminus S^*}\pi_i(M_k) = \|\pi(M_k)\| = 1$, we conclude that for any $k \geq k^*$,

$$\|\pi_i(M_{k+1}) - \pi_i(M_k)\| = 2\sum_{i\in S^*}[\pi_i(M_{k+1}) - \pi_i(M_k)].$$

Therefore, we have for any $\ell \geq k^*$,

$$\sum_{k=k^*}^{\ell} \|\pi_i(M_{k+1}) - \pi_i(M_k)\| = 2\sum_{i\in S^*}[\pi_i(M_{\ell+1}) - \pi_i(M_{k^*})] \leq 2\sum_{i\in S^*}\pi_i(M_{\ell+1}) \leq 2.$$

Q.E.D.

By Theorem V.4.3. of Isaacson and Madsen [27] we have,

Theorem 2.5.1 *The Markov chain $\{X_k\}$ generated by the Stochastic Comparison Algorithm with M_k taking $M_k = \lfloor c \log_\sigma(k + k_0) \rfloor$ is strongly ergodic. Furthermore,*

$$\begin{array}{ll} (1) & \lim_{k\to\infty} \sup_{x_0} \|x(\ell,k) - e^*\| = 0 \\ (2) & \lim_{k\to\infty} \Pr[X_k \in S^*] = 1, \end{array}$$

where $x(\ell,k) = x_0 P(\ell,k) = x_0 \prod_{i=\ell}^{k-1} P(M_i)$, x_0 is the initial probability vector, e^* is an optimal probability vector and c, σ, k_0 are some constants.

Proof. By Lemma 2.5.1 of this report and Theorem V.4.3 in [27], the Markov chain, $\{X_k\}$, is strongly ergodic. Conclusions (1) and (2) are true.

Q.E.D.

Remark In the SC algorithm, we need to decide whether to stay in the current configuration or to move to the candidate one, based on the M_k pairs of samples drawn from the current and the candidate configurations. There are many comparison schemes one can use for making such a decision.

2.6 Numerical Examples

2.6.1 A Testbed System

We designed a testbed system with 1,000,000 configurations. To generate the objective function, we divided the interval [10.0,110.0] into five subintervals each of duration 20.0. Then we generated, $p_\ell\%$, $\ell = 1, ..., 5$, from the total configurations with objective function value uniformly distributed in subinterval ℓ. We consider a minimization problem. To make the search more difficult, we allocate fewer points in the first interval ("good interval") than in the others. The parameters used are shown in Table 2.6.1.

ℓ	1	2	3	4	5
	[10,30]	[30,50]	[50,70]	[70,90]	[90,110]
p_ℓ	3	37	30	15	15

Table 2.1: Percentage of objective function values that fall in each interval

Each sample of objective function for configuration i, is generated according to $H(i) = g(i) + W_i$, where W_i should reflect the behavior of the simulator. In these experiments, we take $W_i \sim \text{unif}[-a/2, a/2]$, $\forall i \in S$.

2.6.2 Test of SC and SR algorithms

The convergence of both SC and SR algorithms are proved with the requirement that the number of tests increases logarithmically. However in practice, the log function increases too fast during the first few iterations. So in the experiment, we compare the two algorithms using a linear sequence. Of course, this will not guarantee the convergence to the optimum. Our purpose here is try to demonstrate that the SC algorithm could have superior performance in practical situations.

The initial configuration X_0 is randomly selected. Our experiments do not indicate any significant difference with different starting configuration. For the testing sequence, we take the linear sequence $M_k = 1 + \lfloor k/500 \rfloor$. It approximates the logarithm sequence within the range of k that we used in the experiment. The generating function is given by $R(i,j) = 1/999999,\ \forall i,j \in S, i \neq j$.

We did the experiments for different noise levels by letting a=0, 10, 20, 30, 40, 50, 60, 70. In the figures, we show the optimization trajectories of SC. To save space, we only present the results for $a = 10, 40, 70$ in Figures 2.1, 2.2 and 2.3.

It can be seen that, even when $a = 70$, the algorithm virtually settles down in about 2200 iterations and the performance is very good.

The SC algorithm has been successfully applied to a realistic computer configuration design problem and a large scale transportation scheduling problem. In the following we present some experiments for the SR algorithm. We emphasize that in general it is hard to "compare" the performance of different optimization algorithms. The particular structure of the problem and the choice of parameters are often crucial. We also emphasize that if there is good neighborhood structure available SR could easily perform better. We argue that in some cases, typically in computer reconfiguration practice, it is hard to find suitable neighborhood structure and it is in these cases SC might be a good choice.

In experiments with SR algorithm, we use the same M_k sequence and choose the stochastic ruler $\Theta(a,b) \sim \text{unif}[0,120]$. The neighbor set $N(i)$ is defined as follows. We put the one million configurations on a circle with equal distance between them. For each configuration, the left one hundred (100) configurations and the right one hundred (100) configurations are chosen as its neighbor set. During the optimization, each neighbor is chosen with equal probability from its neighbor's set of currently visited configurations. Figures 2.4, 2.5 and 2.6 are the optimization results of SR algorithm for $a = 10, 40, 70$ respectively. We also did SR algorithm with "open neighborhood structure" for comparison. Figures 2.7, 2.8 and 2.9 are the optimization results of the SR algorithm with open neighborhood structure for $a = 10, 40, 70$ respectively.

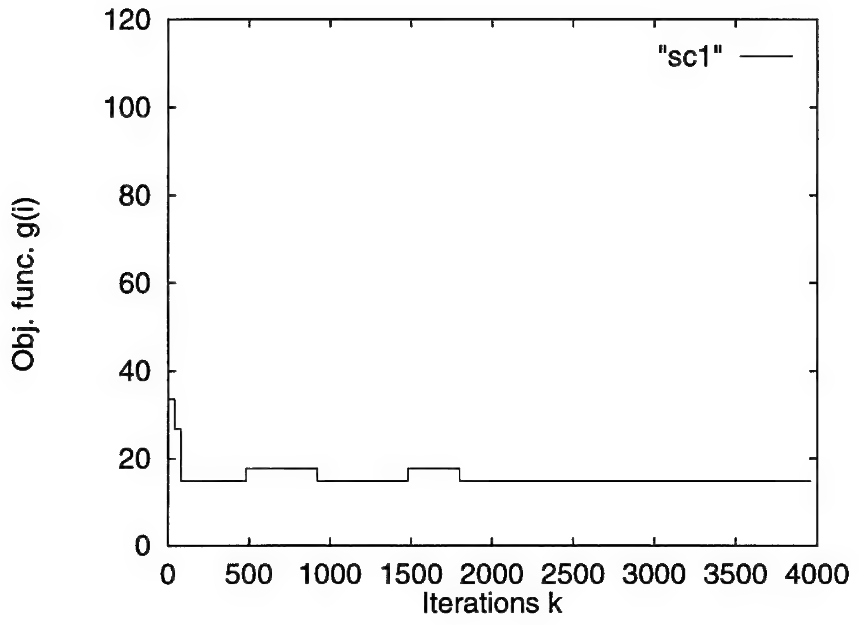


Figure 2.1: Optimization trajectory of SC ($a = 10$)

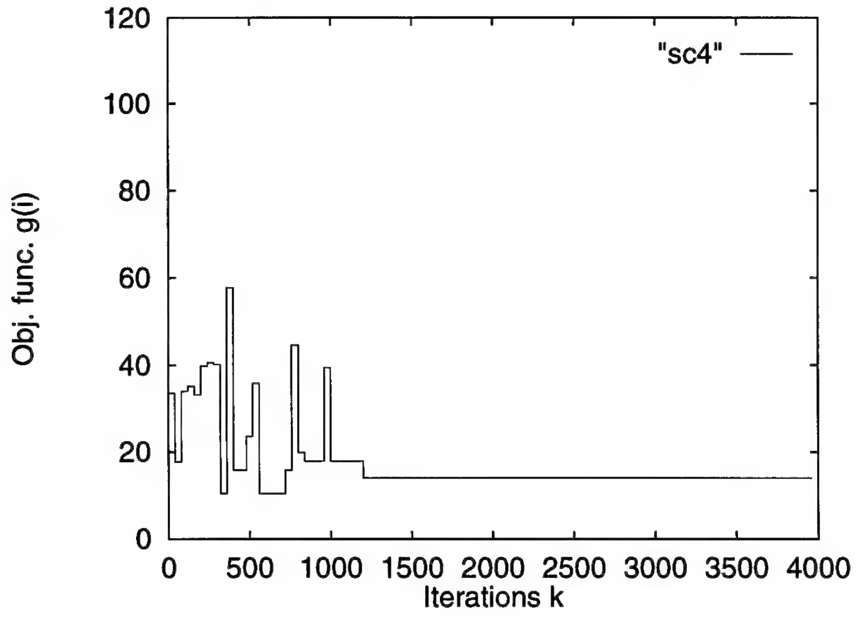


Figure 2.2: Optimization trajectory of SC ($a = 40$)

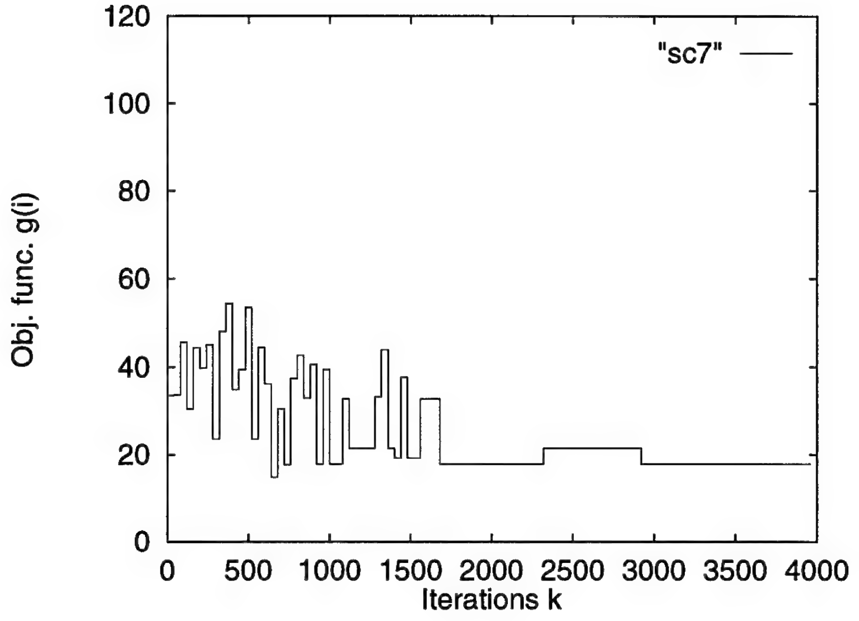


Figure 2.3: Optimization trajectory of SC ($a = 70$)

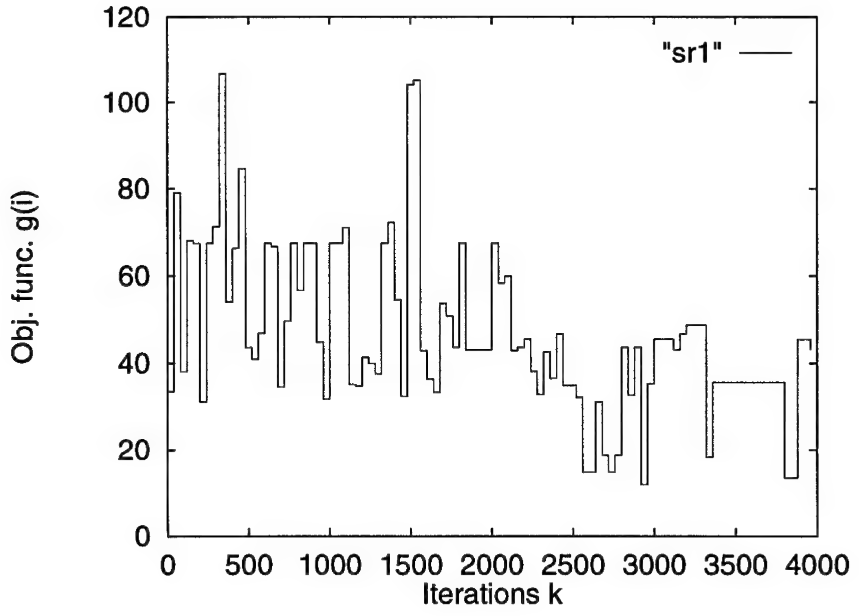


Figure 2.4: Optimization trajectory of SR ($a = 10$)

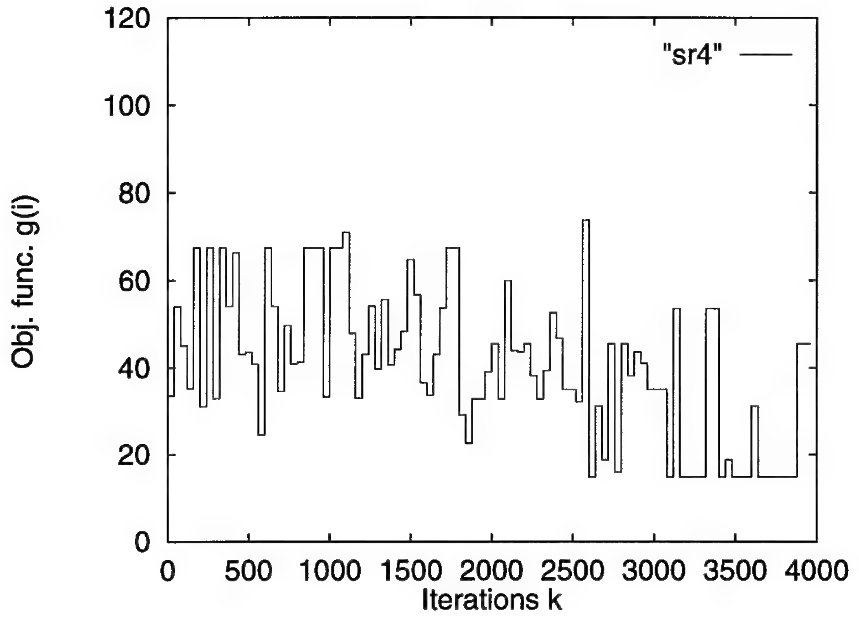


Figure 2.5: Optimization trajectory of SR ($a = 40$)

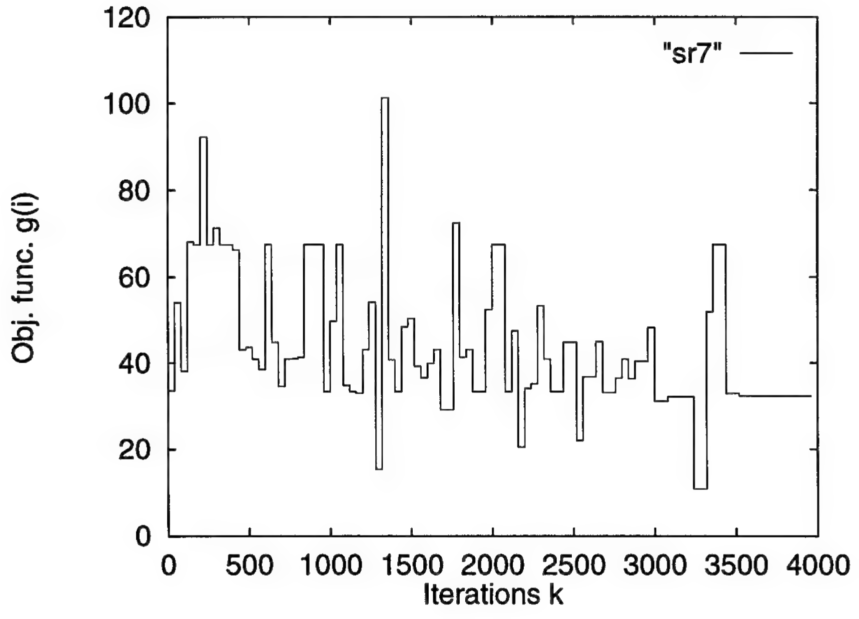


Figure 2.6: Optimization trajectory of SR ($a = 70$)

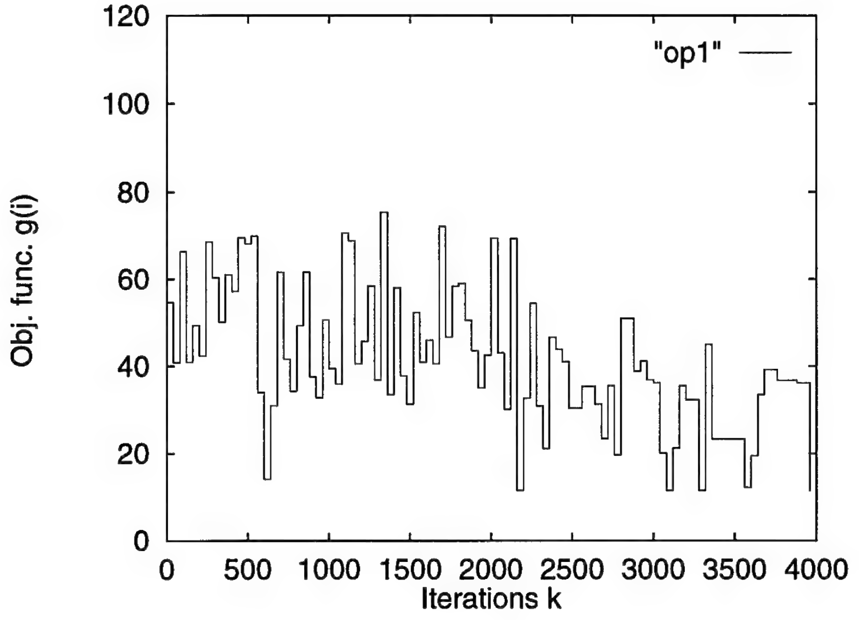


Figure 2.7: Optimization trajectory of SR with open neighborhood ($a = 10$)

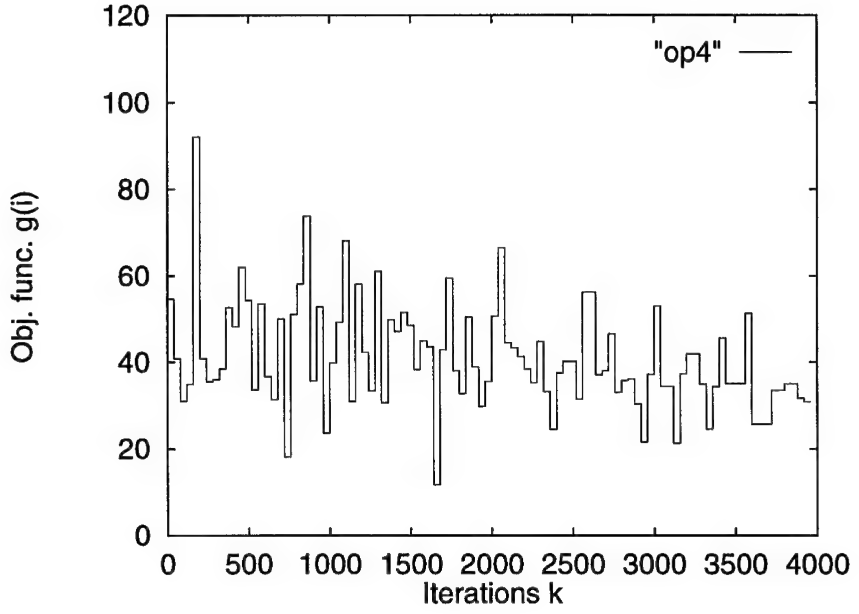


Figure 2.8: Optimization trajectory of SR with open neighborhood ($a = 40$)

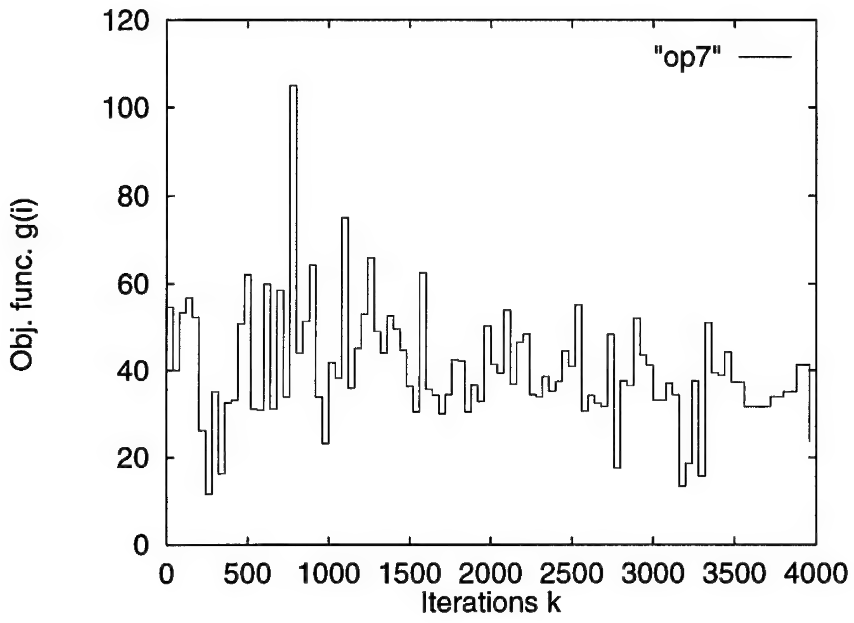


Figure 2.9: Optimization trajectory of SR with open neighborhood ($a = 70$)

2.7 Conclusions

Unlike the SA and SR algorithm, the SC algorithm described in this report does not introduce the neighborhood structure. This is important because in many discrete optimization problems encountered in computer and communication network design, it is difficult to find a natural neighborhood structure. Hence, in many cases, it might be more practical to apply the SC type of algorithms, instead of spending a lot of effort to find a good neighborhood structure before the optimization procedure.

Chapter 3

Window-Based Surveillance Strategies

3.1 Introduction

As computer networks increase in sophistication and provide integrated service to multiple traffic classes (each with its own performance requirements), the problem of network management increases in importance [5]. The network needs to make flow-control, routing, and reconfiguration decisions based on the prevailing traffic on its various channels. Such decisions rely on the accuracy with which the network manager can estimate the prevailing traffic. One approach to estimating traffic intensity is to use window-based surveillance techniques. That is, the traffic on the network is estimated after the network has been monitored over a given window, or duration of time. The issue of the optimum window duration has not been adequately addressed in the literature. This issue increases in importance when the traffic intensity is not constant, but is changing.

Bandwidth is allocated assuming that the traffic pattern is reasonably stable. However, in practice, it is possible for overloads to occur as a result of failures in part of the network, or due to the application. When this happens, the network manager must react by reassigning bandwidth (by, for example, reassigning paths to different circuits) to ensure that the network continues to provide required grades of service. One application where this can happen is a telephone network, using a dynamic, nonhierarchical routing strategy [1, 2, 5]. In [5], for example, the implicit assumption is that the traffic rates stay constant over the window duration.

Another application is the reconfiguration of distributed systems. A distributed system can be made reconfigurable by suitably deploying some switches. Changing the switch settings changes the network structure. It is possible to design strategies for system reconfiguration so that it fits the prevailing traffic pattern [6].

The problem we address in this chapter can be stated as follows. Suppose we wish to determine the traffic in a distributed system or computer network. We can do this by surveillance, i.e., by observing the number of packets over some *window* (i.e., duration of time), and then making the estimate. Standard methods of statistical inference can be used to make the estimate [3, 7]. However, we concentrate in this chapter, on a simpler estimation technique which takes less time (we should keep in mind that the estimation overhead is part of the system control overhead, which should be kept low). The estimate is made by just dividing the number of packets observed by the size of the window.

How large should the window size be to obtain a good estimate of the traffic rate at the end of the window? This problem is not difficult if the traffic rate does not change. If, for example, the traffic is Poisson with constant rate λ, the longer the window, the better will be the estimate of λ. However, if the rate is not fixed, but is changing with time (i.e., it behaves as a nonhomogeneous Poisson process), the problem becomes more interesting. There is a tradeoff: since the rate changes with time, a longer window means that outdated information (from the earlier portion of the window) is being allowed to pollute the estimate. On the other hand, if the window is very short, the probability will be that we have not collected enough statistics to make an accurate estimate of the traffic rate. In this chapter, we study this tradeoff.

Section 2, which contains analysis and numerical results for windows of fixed duration, is the main result of this chapter. In Section 3, we briefly consider windows of a fixed number of packet arrivals, and conclude with Section 4.

3.2 Fixed Duration Window

We assume a stochastic arrival process which can be adequately modelled as a non-homogeneous Poisson process with rate $\lambda(t)$ [8]. Such a process is amenable to analysis without being unduly restrictive: by defining $\lambda(t)$ suitably, we can represent almost any arrival pattern seen in practice.

If n arrivals are seen over a window of duration s covering the interval $[0, s)$ (denote this event by $P(n, s)$) the function $g(n, s) = n/s$ is used to estimate the rate $\lambda(s)$. The objective is to study the expectation, **E**, and variance, **V**, of the estimation error, namely $E[|g(n, s) - \lambda(s)|]$ and $\text{Var}[|g(n, s) - \lambda(s)]$, respectively for various types of function $\lambda(t)$. No restrictions are placed on the form of the function, $\lambda(t)$.

We have

$$\begin{aligned}
\mathbf{E} = E\left[\left|\frac{n}{s} - \lambda(s)\right|\right] &= \sum_{n<s\lambda(s)} \left(\lambda(s) - \frac{n}{s}\right) P(n,s) + \sum_{n>s\lambda(s)} \left(\frac{n}{s} - \lambda(s)\right) P(n,s) \\
&= \sum_{n<s\lambda(s)} 2\left(\lambda(s) - \frac{n}{s}\right) P(n,s) + \sum_{n=0}^{\infty} \left(\frac{n}{s} - \lambda(s)\right) P(n,s) \\
&= \sum_{n<s\lambda(s)} 2\left(\lambda(s) - \frac{n}{s}\right) P(n,s) + \frac{E[n]}{s} - \lambda(s) \qquad (3.1)
\end{aligned}$$

$$\begin{aligned}
\mathbf{V} = Var\left[\left|\frac{n}{s} - \lambda(s)\right|\right] &= E\left[\left(\frac{n}{s} - \lambda(s)\right)^2\right] - E^2\left[\left|\frac{n}{s} - \lambda(s)\right|\right] \\
&= E\left[\frac{n^2}{s^2} - 2\frac{n\lambda(s)}{s} + (\lambda(s))^2\right] - E^2\left[\left|\frac{n}{s} - \lambda(s)\right|\right] \\
&= \frac{E[n^2]}{s^2} - 2\frac{E[n\lambda(s)]}{s} + E[(\lambda(s))^2] - E^2\left[\left|\frac{n}{s} - \lambda(s)\right|\right] \qquad (3.2)
\end{aligned}$$

We have from elementary probability that $P(n,s) = \exp(-\Lambda(s))\frac{(\Lambda(s))^n}{n!}$ where $\Lambda(s) = \int_0^s \lambda(t)dt$. In addition, we have $E[n] = \Lambda(s)$, and $E[n^2] = \Lambda(s)(\Lambda(s)+1)$.

The expectation of the estimation error, $\mathbf{E}$, is a discontinuous function of s: it has jumps at i^+, where $i = 1, 2, \cdots$. It is continuous and differentiable in the intervals $(i, i+1]$.

Unless otherwise noted, in our numerical examples, we will set (without loss of generality) $\lambda(s) = 1$, i.e., the unit of time is defined to be the reciprocal of the arrival rate at the end of the window. Figure 3.1 displays how $\mathbf{E}$ and $\mathbf{V}$ (generically described in the figure as "Estimate Quality") behave when $\lambda(t)$ is constant with respect to time. We have plotted data points at integral values of window size, s. The shape of the curves is encouraging because it indicates that most of the increase in estimation accuracy occurs when the window is increased from 0 to only about 100. Beyond that point, the curve flattens out and very little improvement is to be had for both the expectation and the variance of the inaccuracy. This suggests that even if $\lambda(t)$ is not constant, but varies with time, so long as it does not change appreciably over about 100 interarrival periods, we can make a good estimate.

Let us now turn to what happens when $\lambda(t)$ varies with time. In Figure 3.2 we plot $\mathbf{E}$ and $\mathbf{V}$ for the case where $\lambda(t)$ varies linearly with time: $\lambda(t) = a + bt$. The plots are normalized, so that $\lambda(s) = 1$ (recall that the window ends at time s). As one would expect, $\mathbf{E}$ drops as the window size is increased from a small value. Beyond a certain point, however, the effects of outdated information begin to outweigh the smoothing effects of a larger window, and $\mathbf{E}$ rises again. The greater the value of b, the earlier this point occurs, and the sharper is the rise of the curve beyond it. The variance monotonically declines with increasing window sizes since $\lambda(t)$ is deterministic.

In Figure 3.3, we plot the optimal window sizes (i.e., window for which the expectation of the estimate error is minimized), together with the expectation of the estimation error. The optimal window size decreases monotonically, and **E** increases monotonically, as b increases. For example, if we want an expected estimation error of less than 20%, the rate of change of $\lambda(t)$ must be less than about 1%.

The above examples have all assumed $b > 0$. Similar results hold for $b < 0$.

If $\lambda(t)$ depends on second and higher-order powers of t, it is likely to be changing so fast that the simple estimation approach studied here will not work well. To show this, let us consider the case $\lambda(t) = a + bt^x$. As before, we set $\lambda(s) = 1$ to define the unit of time. Figure 3.4 displays the expectation of the estimation error: this indicates that for such fast-changing $\lambda(t)$, we need more sophisticated techniques.

In all these examples, we have considered specific functions for $\lambda(t)$: we now provide a more general discussion. In Figure 3.5, we plot the expected estimation error for various displacements and windows sizes, assuming that $\Lambda(s) = s(1+\delta)$ for various values of *relative displacement*, δ. The relative displacement accounts for variations in $\lambda(t)$ over the window: recall that we have set $\lambda(s) = 1$.

The expected estimation error reflects two factors: the relative displacement, and the effects of stochastic variations over the window. The latter dominates when the window size is small, and the latter when it is large. In all cases, the expected estimation error cannot possibly be less than the value of the relative displacement (which is, of course, determined by the trajectory of $\lambda(t)$). Beyond a certain value of s, the relative displacement drowns out the effects of stochastic variations over the window. This is expressed in Figure 3.5 by the saturation point, which is defined as the smallest value of window size for which the expected estimation error is no more than 10% greater than the relative displacement.

3.3 Fixed Arrivals Window

Suppose that instead of defining a window as a fixed interval of time, we define it by the number of arrivals. That is, to estimate the arrival rate at the end of a window, we simply determine how long it takes to have n^* arrivals (for some given n^*), and divide the duration of the window by n to estimate $\lambda(t)$ at the end of the window.

Let τ_n denote the time for n arrivals. Then, we can immediately write

$$P[\tau_n \leq x|\lambda(t)] = 1 - \sum_{i=0}^{n-1} e^{-\Lambda(x)} \frac{\Lambda^i(x)}{i!} \tag{3.3}$$

which is, of course, the distribution time of an Erlang process with time-varying arrival rates. Let f_{τ_n} denote the associated density function, i.e.,

$$f_{\tau_n} = \frac{dP[\tau_n \leq x|\lambda(t)]}{dx} = \lambda(x)e^{-\Lambda(x)}\frac{\Lambda^{n-1}(x)}{(n-1)!}.$$

Then, the expectation and variance of the estimation error can be computed numerically from their definitions.

This window behaves similarly to the fixed-time window. In Figure 3.6, we plot the expectation of the error estimate as a function of the size, n, of the window for $\lambda(t) = a + bt$.

3.4 Conclusions

As adaptive control and reconfiguration become more important to distributed systems, telephone, and computer networks, it is important to develop techniques for the system to assess the loading on its various links. This can be done by carrying out surveillance of the network. While many more sophisticated approaches, based on statistical inference are possible, our purpose was to study under what conditions a very simple technique – which imposes very little overhead – would work well.

In this chapter, we have discussed how the window can be chosen over which the traffic is to be monitored. Our results provide guidelines to users for choosing the window size, as a function of the rate of change of the traffic intensity. They indicate that this simple technique is sufficient to obtain good estimates of the traffic intensity parameter, $\lambda(t)$, so long as the rate of change is less than about 2%. Traffic intensities that change much faster than this may indicate one of the following three possibilities: *(a)* a mode change (in which a class of users is switching on or off) which can be detected by looking for abrupt changes in the intensity estimates over successive windows, *(b)* a change according to some law, in which case the coefficients of that law (e.g., a, b in our examples) will have to be measured using a window approach similar to that described here, or *(c)* such chaotic conditions that the traffic intensity parameter at one instant tells us little about the future trajectory of the parameter. In this last case, there is little use in making an estimate of the intensity.

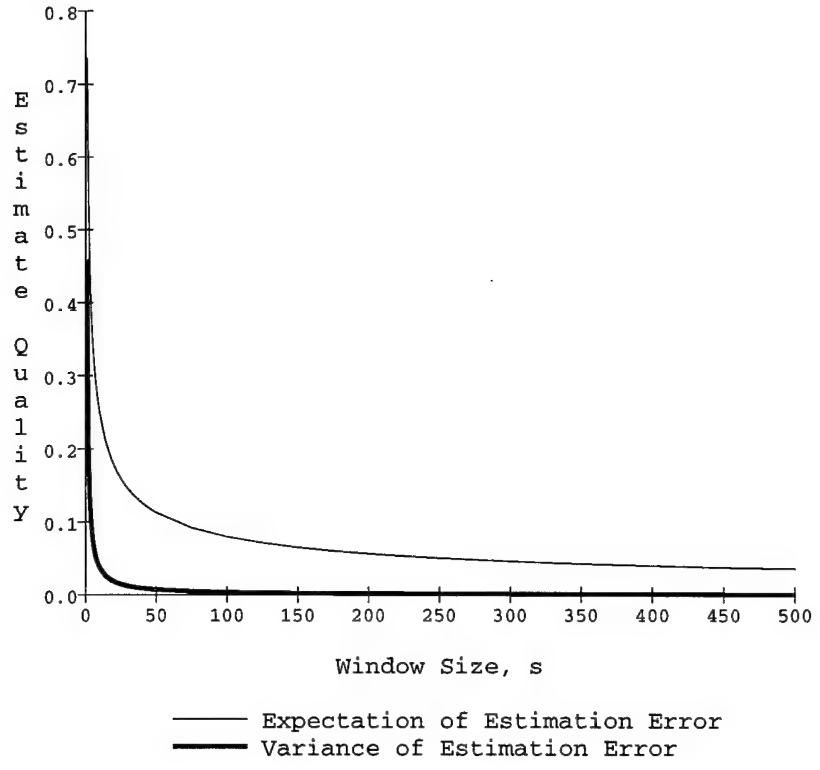
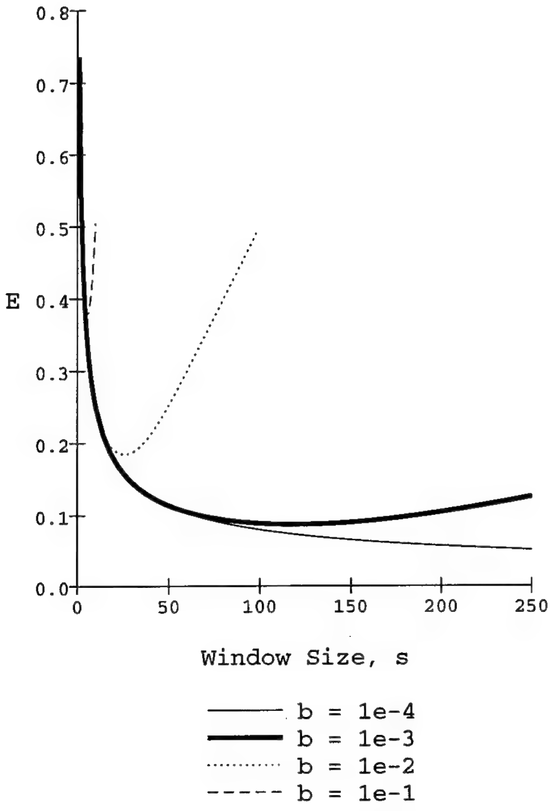


Figure 3.1: Constant Arrival Rate



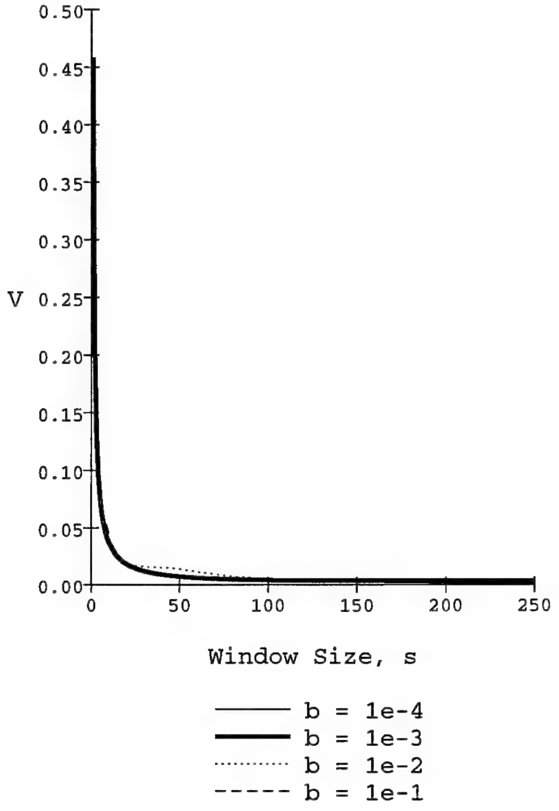


Figure 3.2: Expection and Variance of the Estimate Error for Various Window Sizes

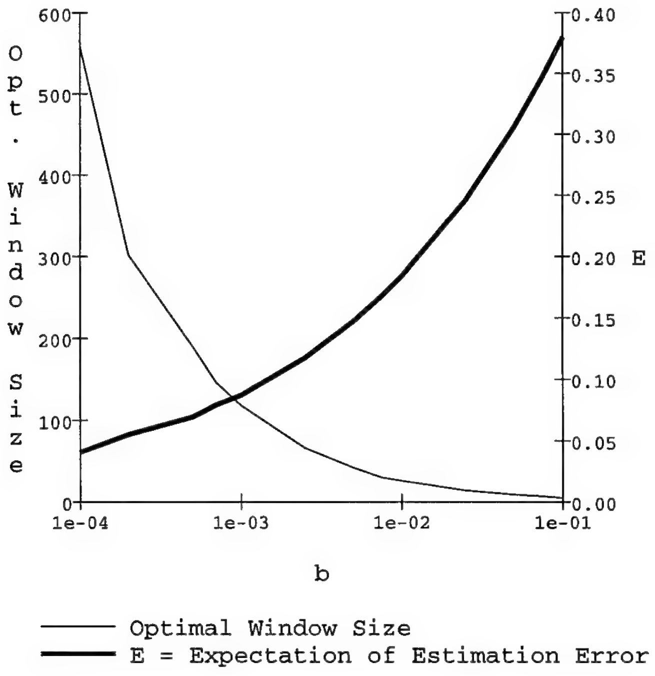


Figure 3.3: Optimal Estimates

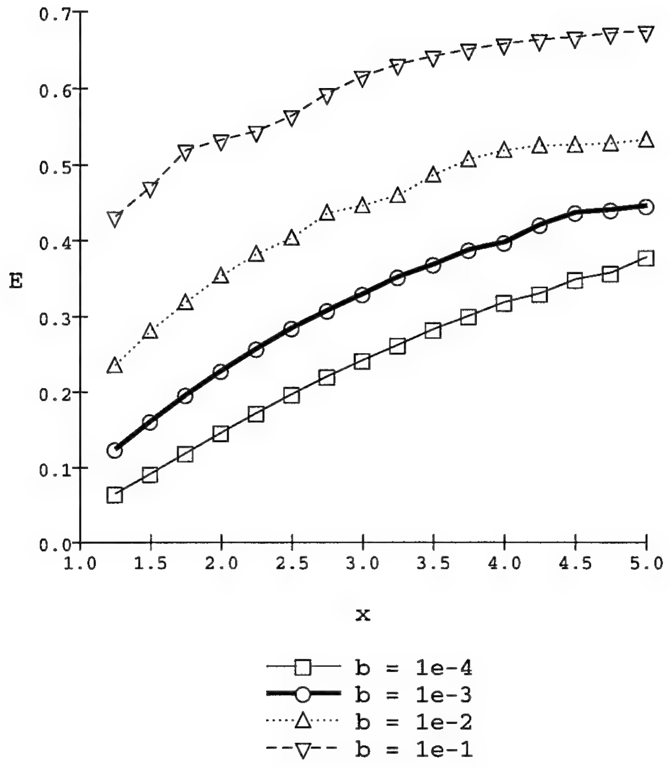
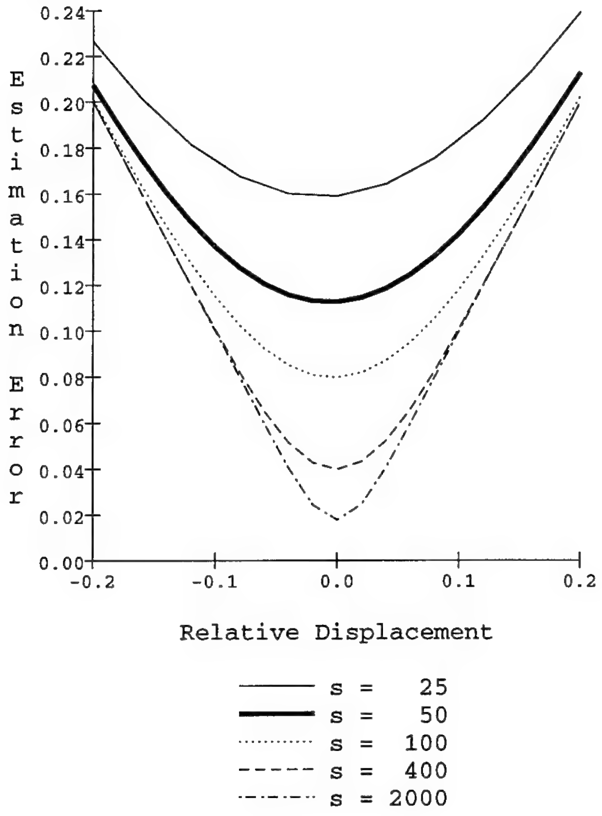


Figure 3.4: Estimation Error for $\lambda(t) = a + bt^x$



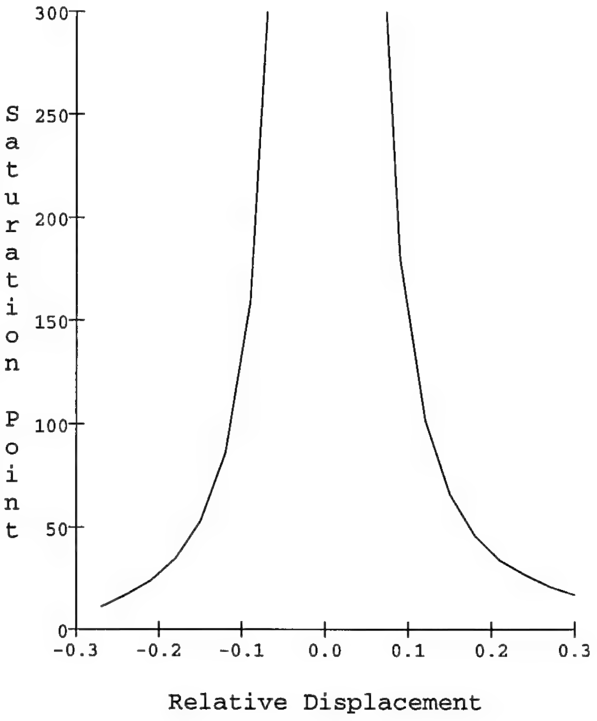


Figure 3.5: Expected Estimation Error and Saturation Point for Various Displacements

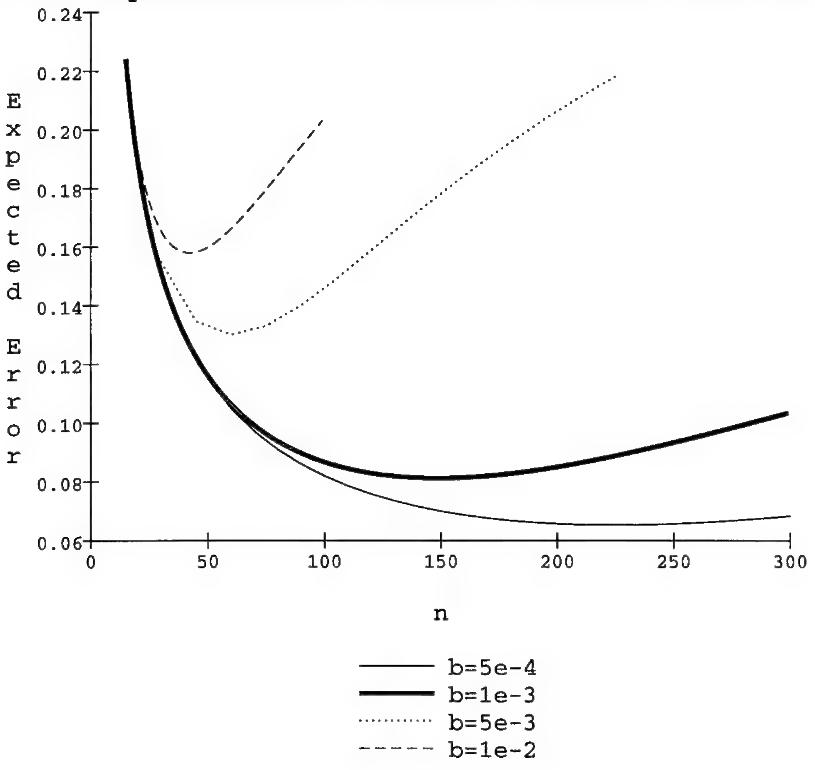


Figure 3.6: Expected Estimation Error for a Fixed-Number Window

Chapter 4

Reconfiguration of Ring Networks

4.1 Introduction

Recent developments in technology have made networks of workstations and personal computers an attractive alternative to mainframe computers for many applications. Such networks offer potential advantages in terms of flexibility: computing power can be incrementally added or removed to suit changing demand; fault-tolerance: in a well-designed distributed system, the failure of a single node will not cause the entire system to collapse; and responsiveness: each node serves a small subset of users, and it is only if shared resources are required that the loading on the rest of the system affects response time. Anecdotal evidence suggests also that such distributed systems are less expensive than their mainframe counterparts.

Factors that determine the power of such distributed systems are the speed of the processors and memory, the operating system, and the structure of the local-area network. The contribution of each of these factors depends on the traffic patterns. In this chapter, we concentrate on the interconnection structure (also called configuration) of ring networks and their adaptive reconfiguration to match prevailing traffic patterns.

Many authors have looked at reconfiguration in multiple-processor systems. Research has focussed on two areas: the reconfiguration of parallel systems to *(a)* counter the effects of the failure of individual components [47, 51, 54, 56, 60, 61] and *(b)* match the structure of a parallel algorithm [55, 57, 59, 60].

In both these areas, the target structure that the system is to configure to is known in advance: in the first case, the system responds to failure by trying to recover its previous structure as far as possible; in the second case, the system tries to match the ***known*** structure of a program. The type of problems that we are considering, on the other hand, require the system to *determine* the best configuration or structure (with respect to some performance

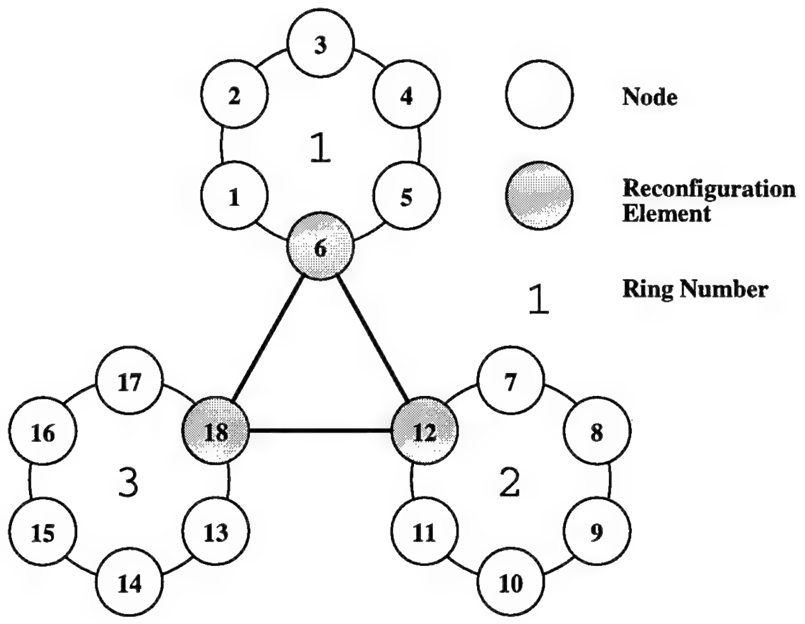


Figure 4.1: The Three Rings Interconnected by Reconfiguration Elements.

functional) before reconfiguring.

We can use the structure in Figure 4.1 to illustrate the significant improvement made possible by adaptive reconfiguration. The structure consists of three token rings, interconnected as shown in the figure. When node i gets a token, it checks its data queue. If the queue is empty, the token is passed to the next node j. If the queue is not empty, the node will transmit all the data frames queued before it releases the token, i.e., this is an exhaustive service protocol. The rings are interconnected by 3 *reconfiguration elements* (RE's), which will be discussed in detail in the next section. The RE's have two modes. In the *separated* mode, data transmitted from one cluster to the other are first buffered at the RE node, and then sent to the next cluster. In the *unified* mode the RE is transparent, i.e., data are forwarded as in a regular node, and the pair of rings that are connected by that RE coalesce into a single ring. Figure 4.2 depicts the various configurations.

The traffic patterns are represented by the matrices D_a and D_b, with $D_x(i,j)$ being the fraction of traffic from ring i to ring j, $i,j = 1,..,3$ in pattern $x = a,b$. The system parameters and traffic patterns and the performance of each configuration (obtained by simulation of 100,000 packets) are shown in Table 4.1. For traffic pattern D_a, Configuration 4 is the best and Configurations 5,6, and 7 have the worst mean packet delays (MPD). There is about a 22-fold improvement in the mean packet delay for Configuration 4 over Configurations 5, 6, and 7. Of course, which configuration is the best and which is the worst depends on the traffic pattern: for pattern D_b, Configuration 4, which was the best for D_a has a mean delay more than twice the delay for the best configuration (i.e., Configuration 1) for D_b. The purpose of this experiment is to drive home the point that a significant potential

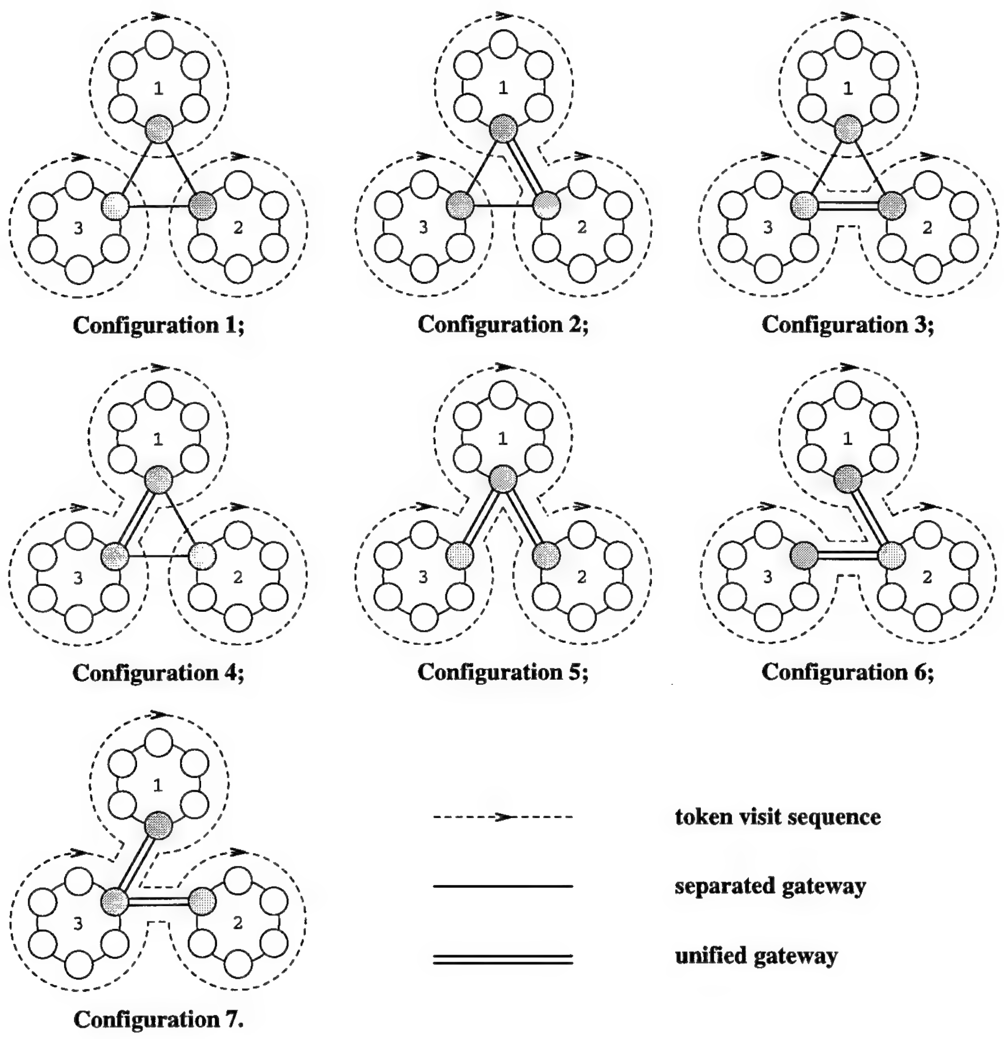


Figure 4.2: All Configurations for Three-Ring System.

exists for performance improvement by causing the network to respond to changes in traffic patterns by reconfiguring itself appropriately.

In the next section, we discuss why locality of message destinations results in hierarchical configurations. We show how reconfiguration can be achieved by suitably fitting the network with reconfiguration elements. Following that, we argue that the reconfiguration problem is one of discrete optimization *with estimation* (DOE). That is, analytical expressions do not exist for the system performance as a function of its configuration, and simulation must be used to estimate it. Traditional techniques are very sensitive to the accuracy of the simulation: in Section 3, we propose to use the recently-introduced *Stochastic Ruler* algorithm which is more robust. The relative insensitivity of this algorithm to the variance of the simulation makes it suitable for use in DOE problems. We conclude with a detailed numerical example of the use of this algorithm.

Configuration	MPD_a	MPD_b
1	1.700	1.920
2	4.460	3.415
3	4.551	3.969
4	1.425	4.194
5	32.075	31.990
6	32.163	32.078
7	32.163	32.078

Channel Bandwidth = 4 Mbps; Total Traffic = 2.7 Mbps;
Propagation Delay per Ring = 0.12 ms; Frame Length = 2000 bits

$$D_a = \begin{pmatrix} .05 & .05 & .90 \\ .10 & .80 & .10 \\ .90 & .05 & .05 \end{pmatrix} \quad D_b = \begin{pmatrix} .10 & .85 & .05 \\ .02 & .08 & .90 \\ .70 & .10 & .20 \end{pmatrix}$$

Table 4.1: Mean Packet Delay (MPD) for a 3-Ring System

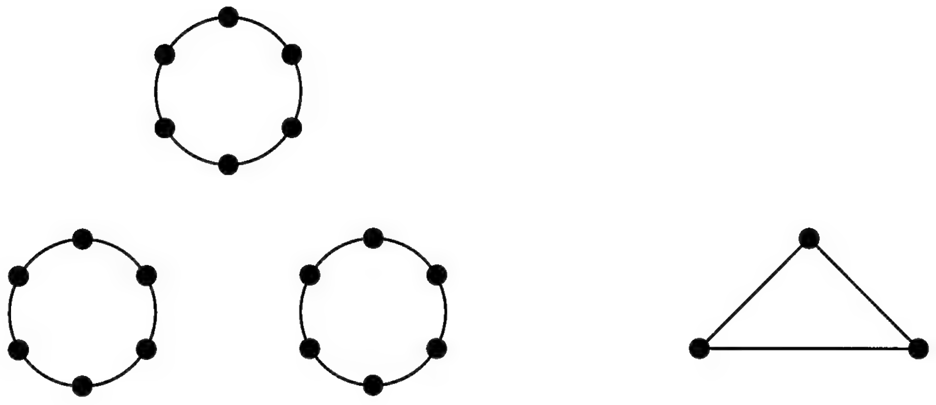
4.2 Hierarchical Configurations

Traffic in large distributed systems tends to exhibit *locality*. That is, the traffic that a given node generates is not uniformly spread amongst all the other nodes, but most of it is to a small subset of nodes. A hierarchical structure can take advantage of such locality by organizing the nodes into a hierarchy of clusters.

Interconnection networks can be depicted in the usual way by means of graphs. If the network is hierarchical, there will be a corresponding hierarchy of graphs. For example, consider the three-ring distributed system example considered in the previous section. This is a two-level hierarchy: the first level hierarchy consists of the three rings, and the second level consists of the interconnection between these three rings. Each node in the second level depicts a ring in the first level. See Figure 4.3.

Reconfiguration will consist of suitably modifying the second-level (or higher-level, if such a level exists) graph.

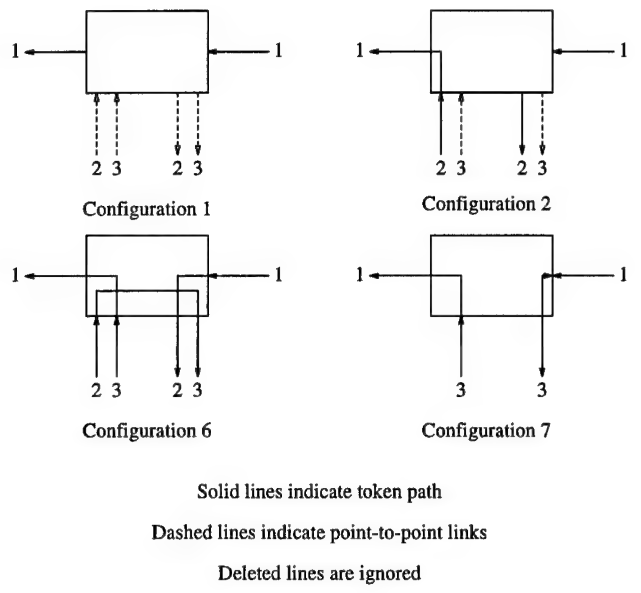
The hardware required for suitably configuring this system, called the *reconfiguration element* (RE), works as follows. When the lower-level nodes are distinct, the RE acts in *separated* (S) mode, as a store-and-forward station. When two nodes coalesce, the RE becomes essentially a "straight-wire" connection, and is said to operate in *unified* (U) mode. In general, an RE consists of buffers, together with a set of input and a set of output lines,



(a) First Level (b) Second Level

Figure 4.3: Graph Depiction of System in Figure 4.1

and can form circuits based on these input and output lines. Consider Figure 4.4.



1
1
2 3
2 3
Configuration 1
1
1
2 3
2 3
Configuration 2
1
1
2 3
2 3
Configuration 6
1
1
3
3
Configuration 7
Solid lines indicate token path
Dashed lines indicate point-to-point links
Deleted lines are ignored

Figure 4.4: Some Settings of the Reconfiguration Element of Ring 1

In Configuration 1, the rings are separate. Packets emanating from ring 1 and destined for rings 2 or 3 are sent to the RE of ring 1. This device then routes messages to rings 2 and 3 along the respective dedicated links. Similarly, the RE of ring 1 receives packets generated in rings 2 and 3, and destined for nodes in ring 1. These are buffered. When the RE of ring 1 receives the ring-1 token, it transmits the packets it has received from the other rings to their ring-1 destinations. The token path is shown in an unbroken line; the point-to-point links are shown as dashed lines.

In Configuration 2, rings 1 and 2 are coalesced with the tokens rotating as shown in Figure 4.2. There is just one token in the combined rings 1 and 2, and the RE passes on the token to the RE of ring 2. When the RE receives this token, it transmits any packets it may have received from ring 3 (which continues to be distinct), before surrendering it to the RE of ring 2. This RE similarly transmits any packets it may have received from ring 3, before passing on the token to the next node in ring 2.

In Configuration 6, the token is passed from ring 1 to ring 2 through the RE, is received from ring 2 and directly passed on to ring 3, and is finally received back from ring 3. Finally, in Configuration 7, rings 1 and 2 have no direct connection, so the connection to/from ring 2 is ignored by the RE.

The process of coalescing two or more rings to form one ring, and vice versa, is carried out by making the RE's reduce and increase the number of tokens, respectively. Consider the transition from Configuration 1 to Configuration 2. When the RE's are given instructions to coalesce rings 1 and 2, they will destroy the tokens of both rings 1 and 2 when they receive them. After both tokens have been destroyed, a new token is generated and sent around the ring made up out of rings 1 and 2. Going the other way is similar. When Configuration 2 is to be turned into Configuration 1, the RE's of rings 1 and 2 turn the link between them to a dedicated store-and-forward link. When the token (of Configuration 2) visits either of these RE's, it is destroyed. Following this, two new tokens are issued, one each for ring 1 and ring 2.

The total number of possible configurations depends on the interconnection structure at the higher levels. If, for example, the second-level structure in our example had been a tree instead of a completely-connected graph, the number of configurations would have been less.

4.3 The Reconfiguration Algorithm

The first step in reconfiguration is to identify the best configuration. This is a difficult optimization problem, which can be formally stated as follows:

> Find the set of configurations S^* such that $g(s) \leq g(s') \; \forall s \in S^* \subset S$ where $S = \{s_1, \ldots, s_k\}$ is the feasible set of system configurations and $g(s)$ is the objective function corresponding to configuration s.

This problem could be solved by using, for example, simulated annealing [49] if there were analytical expressions for the objective function. However, for all but very simple systems, the objective functions (e.g., the average delay) cannot be analytically computed. In fact, all analytical solutions to the token-ring problem that we have seen assume Poisson loading, exponential or fixed packet lengths, and infinite buffer size. Relaxing these constraints renders the problem analytically intractable. Estimates of the objective function, which will reflect the system performance, must thus be dynamically obtained by simulating the system or by experimenting on the system itself. The latter is impractical since in a search-type optimization procedure, this would result in the network switching rapidly from configuration to configuration, disrupting system operation and severely degrading performance. For this reason, we need to choose the former alternative, and simulate the system on one or, if a distributed implementation is desired, several, of the system nodes. Furthermore, to reduce the simulation overhead, the algorithm must be as insensitive as possible to the variance of the performance estimate. Unfortunately, the traditional discrete optimization schemes are very sensitive to the variance of the performance estimate. This means that extremely long simulation runs are necessary to obtain sufficiently accurate estimates for these algorithms.

We therefore use the *Stochastic Ruler* (SR) algorithm, proposed by Yan and Mukai [62] for general discrete optimization problems using Monte Carlo estimates of the objective function. The essence of this algorithm is to compare the estimates of the performance measure at the current configuration with a suitably chosen random variable that is *independent* of the system behavior (called the *stochastic ruler*) to decide whether the system needs to search for a new configuration. The estimate needed for comparison with the stochastic ruler can be rough, since we only need to know *qualitatively* whether this state performs *better* than the stochastic ruler and do not need to know *quantitatively* and by *how much* it performs better. This is along the lines of the ordinal optimization method of [26]. On the other hand, the corresponding Simulated Annealing (SA) algorithm, for example, decides on the direction for the next move of the search based on the *magnitude* of the *difference* between the performance measures of two neighboring configurations. Therefore, for SA, a much more accurate estimate of the objective function will be required. More precisely, consider two configurations s and s' which are to be compared. Let the actual mean of the objective function at configuration x be given by $\mathcal{P}(x)$. Then, if $|\mathcal{P}(s)-\mathcal{P}(s')|$ is large, even a relatively poor estimate of $\mathcal{P}(s)$ and $\mathcal{P}(s')$ will suffice for the SR algorithm to guess correctly which configuration provides better performance. If, on the other hand, $|\mathcal{P}(s)-\mathcal{P}(s')|$ is very small, we have a large probability of making an incorrect guess. *However, because the difference in objective function is so small, there is little penalty associated with making a mistake between the two configurations.*

In other words, the SR algorithm is robust where it matters: in comparing between configurations which differ greatly in performance. When it compares between configurations which are close in performance, an erroneous guess does not matter very much.

The original optimization problem, presented above, required us to find the global optimum set of configurations

$$S^* = \{s \in S | g(s) \leq g(s') \; \forall s' \in S\} \tag{4.1}$$

Let $H(s)$ denote the sample objective function, namely $g(s) = E[H(s)]$ and we use $H(s)$ as the estimate for $g(s)$.

Now, suppose that we know, *a priori*, lower and upper bounds for $g(s)$: let them be $a(s)$ and $b(s)$, respectively. Then, let $a = \inf\{a(s)\}$ and $b = \sup\{b(s)\}$, and $\Theta(a, b)$ denote a random variable distributed over the interval $[a, b]$. $\Theta(a, b)$ is known as a *stochastic ruler* [62]. We can create an alternative optimization problem as follows.

> Find $\Sigma^* = \{s \in S | P(s, a, b) \geq P(s', a, b) \ \forall s' \in S\}$ where $P(s, a, b) = \text{Prob}\{H(s) \leq \Theta(a, b)\}$.

The SR algorithm solves this alternative optimization problem. In [62] it was shown that the set of configurations Σ^* found by the SR algorithm converges to the set S^* in Eqn. (4.1) for Θ uniformly distributed over (a, b).

We now provide, first an intuitive, and then a formal, description of the SR algorithm.

The SR algorithm starts at a prespecified initial configuration, s_0. For each configuration $s \in S$, we define a set of configurations $N(s) \subset S$ as the set of *neighbors* of s. We then choose one of the configurations in $N(s)$, say s', at random, and draw several sample values $x_1, \ldots, x_n$ of the objective function at configuration s'. These sample values are compared with samples $y_1, \ldots, y_n$ drawn from a uniform distribution over the interval $[a, b]$, where a and b are as defined above. If $x_i \leq y_i \ \forall i = 1, \ldots n$, then the search moves to configuration s', otherwise it stays with the current configuration. Intuitively, this will maximize the probability that a sample value of the objective function is smaller than a sample of the stochastic ruler. Since the ruler is the same random variable for all the configurations, this procedure will eventually minimize the objective function.

The formal description of the algorithm requires some definitions to be stated. Let $R(s, s')$, $s, s' \in S$ be a prespecified probability that, if the current configuration is s, the next configuration to be assessed will be s' with probability $R(s, s')$. We require symmetry in R, i.e., $R(s, s') = R(s', s)$. Finally, we need to define an infinite non-decreasing sequence M_k, which satisfies $M_{k+1} > M_k$ and $M_k \to \infty$ as $k \to \infty$. Now we can describe the SR algorithm.

The Stochastic Ruler Algorithm

Data : N, R, $\{M_k\}$, a, b, $s_0 \in S$.

Step 0: Set $X_0 = s_0$ and $k = 0$

Step 1: Given $X_k = s$, choose a candidate Z_k from $N(s)$ with probability distribution $\text{Prob}[Z_k = s' | X_k = s] = R(s, s'), s' \in N(s)$.

Step 2: Given $Z_k = s'$, set

$$X_{k+1} = \begin{cases} Z_k & \text{with probability } p_{SR}(k), \\ X_k & \text{with probability } (1 - p_{SR}(k)), \end{cases}$$

where $p_{SR}(k) = \{\text{Prob}[H(s') \leq \Theta(a,b)]\}^{M_k} R(s,s')$.

Step 3: Set $k = k + 1$ and go to Step 1.

It has been shown that if $M_k = \lfloor c \log_\beta(k + k_0 + 1) \rfloor$ for some positive real numbers c, β, k_0, the probability that the algorithm has found the optimal configuration is monotonically increasing with k [62].

4.4 Evaluation of the Reconfiguration Algorithm

We provide two illustrations to emphasize the robustness of the SR algorithm with respect to simulation accuracy. In the first, we return to the three-ring system that we introduced in Section 2, with load matrix D_a. Recall that we carried out intensive simulations of the network in each of its configurations as shown in Table 4.1. These numbers indicate that configurations 1 and 4 are the best for that traffic pattern: they are so close in performance that there is little to choose between them.

We ran the SR algorithm on the three-ring system. We take Θ uniformly distributed over $[1, 33]$, $M_k = 1 + \lfloor k/30 \rfloor$, $s_0 = 0$ and each configuration has three (3) neighbors.

In Figures 4.5 and 4.6, we show the optimization trajectory of the algorithm against the number of iterations. To obtain the performance of the network at each configuration, the SR algorithm carries out a simulation[1]. In Figure 4.5, this simulation is run for a total of 1000 packets per iteration, while in Figure 4.6, it is based on only 50 packets. Using only 50 packets gives us no more than a crude estimate. *Nevertheless, even with this crude estimate, the performance of objective function is close to that when it has a better estimate.* In the end, both algorithms oscillate between two configurations (1 and 4) of approximately the same performance. We emphasize that this is for illustrative purposes only: as small a system as this with a total of only seven configurations clearly does not require a sophisticated optimization algorithm.

Our second illustration, by contrast, consists of a system with 1000 configurations, whose objective function ranges from 5 to 55. The objective function associated with each of the configurations is shown in Figure 4.7. We then ran the SR algorithm on this system. We choose Θ uniformly distributed over $[5, 55]$, $M_k = 1 + \lfloor k/40 \rfloor$, $s_0 = 580$ and each configuration

[1]This simulation is distinct from that mentioned in Table 4.1, which was for the accurate estimation of the objective function purposes.

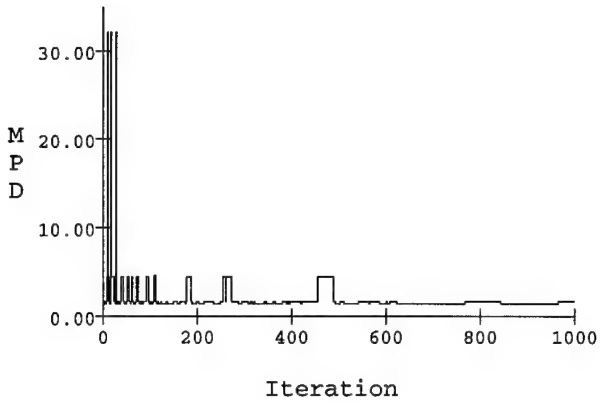


Figure 4.5: 1000 Packets per Simulation Run: Optimization Trajectory of Three-Ring System.

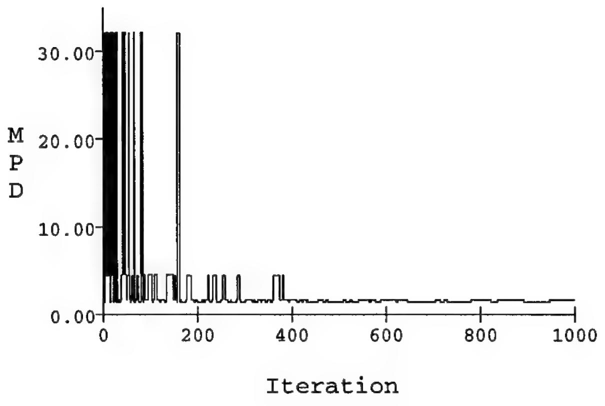


Figure 4.6: 50 Packets per Simulation Run: Optimization Trajectory of Three-Ring System.

has sixty (60) neighbors. The estimate of the SR algorithm is simulated by adding to the objective function in Figure 4.7 some uniform random noise. The variance of the estimate is then the variance of the added random noise. In Figures 4.8 and 4.9, we consider the case where the added noise is uniformly distributed in the interval $[\alpha, \beta]$ for different values of α, β. In Figure 4.8, we display the SR algorithm trajectory for the case $[\alpha, \beta] = [-0.1, 0.1]$, and compare this in Figure 4.9 with the trajectory for the case $[\alpha, \beta] = [-2.4, 2.4]$. The algorithm for the first case converges to a configuration with objective function of approximately 5 (the global-best configuration), while for the second case, with a significantly cruder estimate – with a variance 576 times that of the $[-0.1, 0.1]$ case, it converges to objective function of about 12.5. While such a configuration is not the global optimum, it is still within the best 15% of all the configurations. This is the price that has to be paid for a saving of **96%** of the simulation time over that of the $[-0.1, 0.1]$ case. As with our first illustration, this example has attested to the robustness of the SR algorithm with respect to the variance of the estimate.

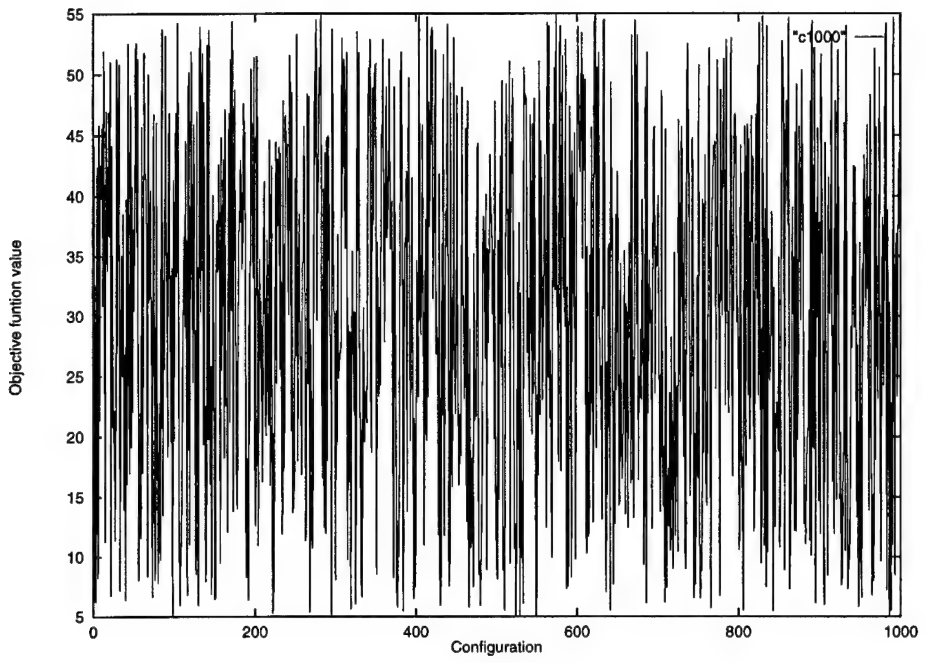


Figure 4.7: Objective Function for the 1000-Configuration System.

4.5 Conclusions

In this chapter, we presented the potential of reconfiguration to improve the performance of networks by dynamically altering their structure to better fit the current traffic pattern. This performance improvement can be significant, and does not require substantial additional hardware.

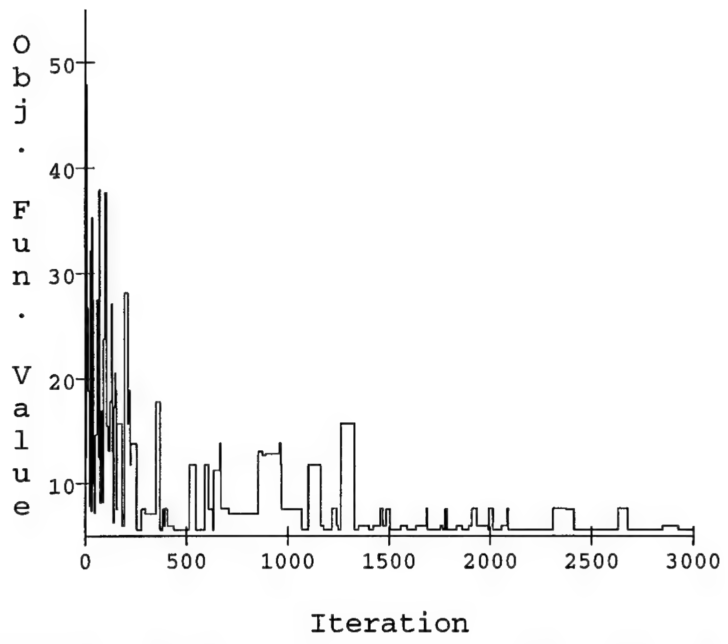


Figure 4.8: Optimization Trajectory of 1000-Configuration System ($[\alpha, \beta]$ = [-0.1,0.1]).

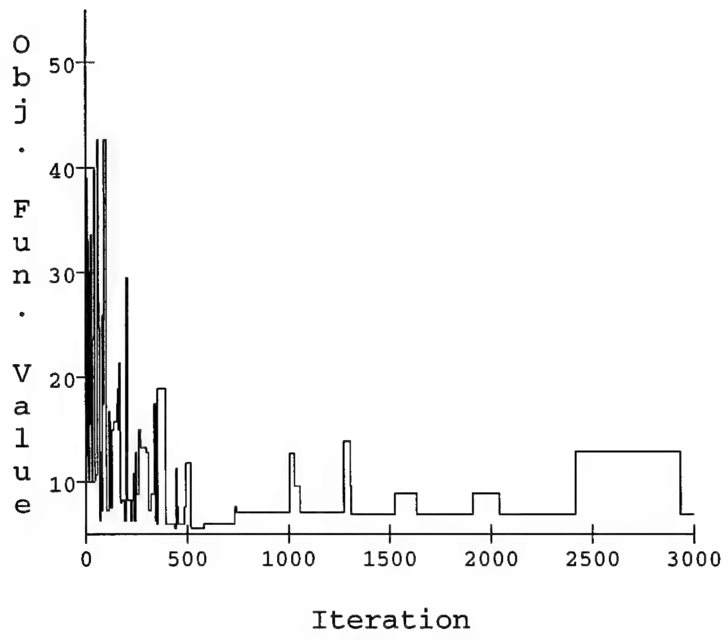


Figure 4.9: Optimization Trajectory of 1000-Configuration System ($[\alpha, \beta]$ = [-2.4,2.4]).

Chapter 5

Reconfigurability in Lightwave Networks

In this chapter we present a new approach to the design of multihop lightwave networks with connectivity patterns that can be dynamically reconfigured. We introduce a practical algorithm that efficiently reconfigures the connectivity diagram by reassigning wavelengths to best fit the current traffic pattern. We implement the stochastic ruler algorithm to obtain the mapping of a regular structure into a WDM star, while optimizing the system performance measures. The results show that this is a practical tool and the resulting structures have a superior system performance.

5.1 Introduction

A typical WDM based local lightwave network uses a passive star coupler as the underlying physical topology [35]. The key property of such WDM based networks is its relative *independence between the logical interconnection pattern and the physical topology.* The logical connectivity among the system nodes is obtained by the assignment of wavelengths to the system transmitters and receivers. Two approaches have been proposed in the literature. In the first approach, the wavelength assignment is performed per packet basis (according to the packets present in the system and the hardware constraints) [36, 37], resulting in a single hop network. However, this approach requires fast tunable optical components and fast pre-transmission coordination among the users who wish to communicate. In the second approach, a quasi-static virtual topology is created in the network, requiring only *fixed* (slowly tunable) transmitters and receivers and resulting in a multihop network [38, 39].

The independence between the *multihop virtual topology* and the underlying physical topology (the WDM passive star) provides a large flexibility in the domain of topological design. The objective functions in the design of virtual topologies are 1) to increase the user accessible bandwidth, 2) to reduce the average delay, and 3) to minimize the hardware cost in terms of the number of transmitters and receivers per node, number of buffers per node and number of wavelengths in the network. In multihop structures, the increase in the user accessible bandwidth and reduction of the average delay can be obtained by designing inherently load balanced structures with a small diameter, or a *small average number of hops*. Moreover, due to the discrepancy in the speed of the optical transmission and the electronic processing, the amount of electronic processing in the network has to be reduced, i.e., simplification of the processing at each intermediate node is required.

Several papers [38, 39, 40] investigated the properties of *regular virtual topologies* and concluded that several major advantages are obtained by selecting certain regular topologies : 1) simplified routing and congestion control procedures can be implemented to reduce the amount of electronic processing required at each intermediate node; and 2) simplified network implementation due to the standard hardware requirement at each node. However, the benefits of these structures may be hindered due to the *unbalanced load* imposed on the regular structure. For example, asymmetric traffic patterns may create bottlenecks, substantially deteriorating the system performance, i.e., increasing the average delay and average queue length. To alleviate this problem and accordingly optimize the system performance, the mapping of the regular structures into the physical structure should *consider the expected traffic patterns in the system.* Moreover, appropriate objective functions should be chosen, so that they will reflect the stochastic behavior of the system under realistic assumptions. The existing approaches of embedding regular topologies on a WDM star do not address this issue. As we will argue in the following sections, incorporating the traffic considerations, can dramatically improve the system performance.

In [41] the authors construct the network virtual topology for a given traffic pattern. The objective function they chose to optimize is the maximum flow on a link in the network. The problem is formulated as a mixed integer optimization problem subject to linear constraints, an approach commonly used for topological design of data networks. However, the min-max performance measure (minimizing the largest flow) does not properly reflect the system performance, i.e., the average packet delay or average queue length.

In this chapter, we present a different formulation and solution to the wavelength assignment problem. We provide the mapping of regular structures into a WDM star, while considering the traffic pattern imposed on the system. This mapping will optimize any objective function that is a function of the stochastic traffic patterns, such as the system throughput, the average delay, the average queue length, etc. We propose to solve the wavelength assignment problem in two phases: the first one is the virtual topology design, and the second one is the mapping of the virtual topology into the physical network. In the first phase we will adopt regular structures as the virtual topology of the WDM star. The use of regular structures will reduce the processing at intermediate nodes, by simplifying the routing and congestion control procedures. In the second phase the regular topology adopted in the first phase, has to be mapped into the network nodes according to the expected traffic demands. As we can see from the wavelength assignment problem with hypercube network structure and hypercube traffic pattern (Table 3.1), there is a large gap between the performance of a random mapping and a traffic tailored one. Notice that the search space for our mapping problem grows as fast as $N!$, leading to the need to develop a heuristic approach. *Moreover – and this is a crucial point – for all but very simple systems, the objective functions (e.g., the average delay) cannot be analytically computed. Estimates of the objective function, which will reflect the system performance, must be dynamically obtained by simulating the system.*

The traditional discrete optimization schemes, such as simulated annealing [42], are sensitive to the variance of the performance estimate. Therefore, extremely long simulation runs are necessary. We thus need a new approach that is less sensitive to the variance of the performance estimate.

We therefore propose to use a new approach, the so-called *Stochastic Ruler* (SR) algorithm, first proposed in [43] for general discrete optimization problems using Monte Carlo estimates of the objective function. The essence of this algorithm is to compare the estimates of the performance measure at the current state with a suitably chosen random variable that is *independent* of the system behavior (called the *stochastic ruler*) to decide whether the system needs to search for a new mapping. The estimate needed for comparison with the stochastic ruler can be rough, since we only need to know *qualitatively* whether this state performs *better* than the stochastic ruler and do not need to know *quantitatively* and by *how much* it performs better. On the other hand, the corresponding Simulated Annealing (SA) algorithm, for example, decides the next move of the search based on the *magnitude* of the *difference* between the performance measures of two neighboring states. Therefore, for SA, a much more accurate estimate of the objective function will be required. In consequence the SR algorithm will be significantly faster.

In Section 2 we describe the model and formally define the wavelength assignment problem. In Section 3 we introduce the Stochastic Ruler algorithm. Several examples are provided in Section 4, and Section 5 concludes the chapter.

5.2 The Wavelength Assignment Problem

We consider a WDM passive star network which interconnects N nodes. Each node has d transmitters and d receivers. Each transmitter/receiver can be tuned to any wavelength in the system. A directed link exists from node i to node j if one of node j receivers receives on one of the wavelengths on which node i transmits. Therefore, the system virtual topology is obtained by proper assignment of wavelengths to the systems' transmitters and receivers.

The directed graph representing the virtual topology is denoted by the *host graph*,

$$G_H =< V_H, E_H > \tag{5.1}$$

$v_i^H \in V_H$ represents a node i in the host architecture, denoted by the host node.

$(v_i^H, vj^H) \in E_H$ represents a directed edge from host node i to host node j.

The packet generation rate of the network is λ. The traffic demand matrix is given by matrix P, with entry $[p_{ij}]_{i,j=1..N}$, denoting the probability that a packet has source i and destination j. This demand matrix can be viewed as the adjacency matrix of a graph, denoted as the *image graph*,

$$G_I =< V_I, E_I, W_I > \tag{5.2}$$

$v_i^I \in V_I$ represents the real node i in the network ($i = 1..N$), denoted by the image node.

$(v_i^I, v_j^I) \in E_I$ represents the existence of a traffic demand ($p_{ij} > 0$) from image node i to image node j.

$w_{(v_i^I, v_j^I)} \in W_I$ represents the average packet transmission rate from image node i to image node j.

$$w_{ij} = \lambda \times p_{ij} \tag{5.3}$$

To solve the wavelength assignment problem we propose a two phase solution: in the first phase we provide the design of the host graph and in the second phase the host graph is mapped into the image graph.

Phase 1: Host Graph Design

We propose to adopt regular topologies, based on the following reasons [40, 41]:

1. Regular structures have simplified routing and congestion control procedures, which will reduce the amount of electronic processing at each node.

2. Simplified network implementation due to the standard hardware requirements at each node.

In Section 4 we apply our algorithm to some regular topologies such as the Hypercube and the Shufflenet.

Phase 2: Mapping of the Host Graph into the Image Graph

In this phase we map the host graph designed in Phase 1 into the image graph defined by the network traffic patterns. We first define the mapping state s_k by a vector

$$s_k = \{(s_k^1, s_k^2, s_k^3, \ldots, s_k^N) |\ s_k^i \in (1, N) \\ \textit{and } \ i \neq j \Rightarrow s_k^i \neq s_k^j\}$$

If $s_k^i = v_j^I$, node v_i^H is mapped into node v_j^I. The mapping s_k determines the network configuration, i.e., the wavelength assignment to the system's transmitters and receivers. $S = \{s_1, \ldots, s_n\}$ represents the mapping space, i.e., the set of feasible mappings.

The Routing Strategy: For obtaining relevant system performance measures, the routing strategy implemented in the simulation should be close to the one used in the real system. Since in optical transmission systems, the time involved in the electronic processing of each packet at the intermediate nodes constitutes a heavy penalty when compared with the transmission delay, simple routing strategies are preferred, such as source routing algorithms. In the examples we provide in a later section we use the shortest path routing.

The Objective Function: The objective function $g(s)$ that we choose to optimize reflects the system performance in face of stochastic demand. For example we can choose to optimize the average packet delay or system throughput. For all but very simple systems, we are unable to obtain an analytical expression for the objective function $g(s)$. Monte Carlo simulations are therefore the main tool to evaluate $g(s)$.

The Wavelength Assignment Problem: Find the global minimum set of mappings

$$S^* = \{s \in S | g(s) \le g(s') \ \forall s' \in S\} \tag{5.4}$$

Usually we use the estimate based on a single sample path of the system evaluation process to evaluate $g(s)$. This estimate is called the sample performance index and is denoted by $H(s)$. We require $E[H(s)] = g(s)$. Accurate estimation of system performance implies long simulation length. Since in a random search algorithm we will have to evaluate the performance index at many states, to obtain accurate estimate for the performance index is in general formidable. In fact, given the temperature control sequence $\{T\}$ and some increasing and convex function $[]^+$, in Simulated Annealing a new state candidate Y_k is accepted as the next state X_{k+1} with probability

$$P_k = \exp^{-\frac{1}{T}[g(Y_k)-g(X_k)]^+} \tag{5.5}$$

which is based on the magnitude of the difference $g(Y_k) - g(X_k)$. However, the difference obtained by $H(Y_k) - H(X_k)$ would be a poor estimate of $g(Y_k) - g(X_k)$ unless we adopt a large simulation effort for this evaluation. This is why we believe that Simulated Annealing is not suitable for our problem. In the next section we provide a detailed description of the SR algorithm and its implementation in the wavelength assignment problem.

5.3 The Stochastic Ruler Optimization Algorithm

We first introduce some notations. Let a, b denote the lower and upper bounds of $H(s)$, $\forall s \in S$, respectively. Let $\theta(a, b)$ be a random variable uniformly distributed in interval $[a, b]$.

We call it the stochastic ruler. Let $P(s,a,b)$ denote the probability that a sample of the index, $H(s)$, is less than the ruler $\theta(a,b)$. Namely $P(s,a,b) = \Pr(H(s) < \Theta(a,b))$. In the SR algorithm we base our decision about whether to move to another state, not on the difference of the performance index value of the current state and a neighboring state, but on the comparison of the sample value of the current state and the fixed stochastic ruler. The later one is just an *ordinal comparison* and therefore should take less simulation effort to obtain, since orders of the random variables are more robust against the variance [44, 45]. To achieve this we have to convert our original optimization problem (4) into an alternative one described as follows.
Find $\Sigma^* = \{s \in S | P(s,a,b) \geq \ P(s',a,b)\ \forall s' \in S\}$

In [43] it is shown that the set of mappings Σ^* found by the SR algorithm converges to the set S^* in (4) under very mild conditions.

The SR algorithm starts at a prespecified initial mapping, s_0. For each mapping $s \in S$, we define a set of mappings $N(s) \subseteq S$ as the set of ***neighbours*** of s. We then choose one of the mappings in $N(s)$, say σ, at random, and draw several sample values $x_1, \ldots, x_n$ of the objective function at mapping σ. These sample values are compared with samples $y_1, \ldots, y_n$ drawn from a uniform distribution over the interval $[a,b]$, where a and b are as defined above. If $x_i \leq y_i\ \forall i = 1, \ldots n$, then the search moves to mapping σ, otherwise it continues from the same mapping. Intuitively, this will maximize the probability that a sample value of the objective function is smaller than a sample of the stochastic ruler. Since the ruler is the same random variable for all the mappings, this procedure will eventually minimize the objective function.

The formal description of the algorithm requires a few more definitions. Recall that $N(s)$ is the set of neighbors of mapping s. Let $R(s,s')$, $s,s' \in S$ be a prespecified probability that, if the current mapping is s, the next mapping to be assessed will be s'. We require symmetry in R, i.e., $R(s,s') = R(s',s)$. Finally, we need to define an infinite non-decreasing sequence M_k, which satisfies $M_k \to \infty$ as $k \to \infty$. We next describe the SR algorithm.

The Stochastic Ruler Algorithm

Data : $N,\ R,\ \{M_k\},\ a,\ b,\ s_0 \in S.$

Step 0: Set $X_0 = s_0$ and $k = 0$

Step 1: Given $X_k = s$, choose a candidate Z_k from $N(s)$ with probability distribution $\text{Prob}[Z_k = s' | X_k = s] = R(s,s'), s' \in N(s)$.

Step 2: Given $Z_k = s'$, set

$$X_{k+1} = \begin{cases} Z_k & \text{with probability } p_{SR}(k), \\ X_k & \text{with probability } (1 - p_{SR}(k)), \end{cases}$$

where $p_{SR}(k) = \{\text{Prob}[H(s') \leq \Theta(a,b)]\}^{M_k}$.

Step 3: Set $k = k + 1$ and go to Step 1.

Step 2 of the above algorithm may be accomplished as follows. Given $Z_k = s'$, draw M_k samples $\{h_i(s')\}_{i=1}^{M_k}$ from $H(s')$. Next draw M_k samples $\{\theta_i\}_{i=1}^{M_k}$ from $\Theta(a,b)$. If $h_i(s') \leq \theta_i$ for $i = 1, 2, ..., M_k$, then set $X_{k+1} = Z_k = s'$; otherwise, set $X_{k+1} = X_k$.

The implementation of this algorithm for the wavelength assignment problem is described in the following subsection.

Implementation of the Stochastic Ruler Algorithm

Since the SR algorithm requires that every state can be reached from any other state and also that $s' \in N(s) \leftrightarrow s \in N(s')$, we define the neighborhood structure in the following way. Let $s = (s^1, s^2, \ldots, s^i, \ldots, s^j, \ldots, s^N)$, then

$$N(s) = \{s' |\ s' = (s^1, \ldots, s^j, \ldots, s^i, \ldots, s^N), \\ \forall\ 1 \leq i, j \leq N \textit{ and } i \neq j\}$$

Therefore the size of $N(s)$ is

$$|N(s)| = \frac{N \times (N-1)}{2}$$

The transition probabilities $R(s, s')$ are uniformly distributed within the neighboring set, i.e.,

$$R(s, s') = \begin{cases} \frac{2}{N \times (N-1)} & \text{if } s' \in N(s) \\ 0 & \text{otherwise} \end{cases}$$

The simulation number control sequence $\{M_k\}$ is nomally chosen as follows (In some experiments in Section 4, we use different $\{M_k\}$ to observe the impact of this control sequence) .

$$M_k = 1 + \lfloor \frac{k}{N \times (N-1)} \rfloor$$

where k is iteration count and $\lfloor\ \rfloor$ is the floor function. It was shown in [43] that if $M_k = \lfloor c \log_\beta(k + k_0 + 1) \rfloor$ for some positive real numbers c, β, k_0, the probability that the algorithm has found the optimal mapping is monotonically increasing with k to 1. However this is too slow for a practical calculation.

The initial state is randomly chosen within the set S. The algorithm will halt in a certain state if M_k reaches a predefined value ϵ.

5.4 Results

In this section we show that the wavelength assignments obtained by the SR algorithm provide substantially lower average delay then random assignments (that do not consider the traffic patterns). We will also provide experiments that determine the quality of the wavelength assignment obtained by the SR algorithm. These experiments evidence that the SR algorithm obtains near optimum results, and has the ability to find the top states from a very large state space.

The assumptions are summarized below:

1. Traffic Pattern: the mean packet arrival rate is represented by λ, and the traffic distribution is represented by matrix P.

2. Physical Parameters of the Network: the packet transmission time per packet hop is deterministic, $\frac{1}{\mu}$, and the buffer at each node is infinite.

3. Objective Function: the mean packet delay obtained from the middle $[\alpha..\beta]$ transmitted packets in the system simulation:

$$F = \frac{1}{\beta - \alpha} \times \sum_{i=\alpha}^{\beta} d_i$$

4. M_k for the SR algorithm:

$$M_k = 1 + \lfloor \frac{k}{\delta} \rfloor$$

where k is the iteration count, δ is the M_k parameter, and $\lfloor\ \rfloor$ is the floor function.

5. Initial State: chosen randomly.

All the experimental runs in this section were executed on a DECStation 5000 running ULTRIX. For the evaluation of the computational complexity of the proposed solution we also provide the CPU time for each run.

Identical Structures Mapping

First we would like to evaluate the assignment obtained by the proposed solution. Obviously one way to assess these results is to carry out an exhaustive search. However, since the search space of the problem, as shown in the previous section, is exponentially growing, such a comparison is infeasible. We therefore propose to use the concept of Identical Structure Mapping. The Identical Structure Mapping(ISM) is obtained when both the Host Graph (i.e. the regular structure) and the Image Graph (i.e. the traffic pattern) have identical structures. Since the structures are identical, the optimum mapping can be found at the structural level, and can be expressed as:

Rule 1: state s in ISM is

$$\begin{cases} \textit{optimum} & \text{if all traffic in traffic pattern} \\ & \text{can be mapped to an one hop} \\ & \text{link in the Host Graph;} \\ \textit{not optimum} & \text{otherwise} \end{cases}$$

Moreover, since the optimum solution in ISM is that all packets have an one-hop route, we can use the M/D/1 queueing model to analytically compute the system performance index. Let λ be the packet arrival rate, $P = [p_{ij}]$ the traffic distribution pattern and $\frac{1}{\mu}$ the transmission time of each packet in each hop. Let also $\rho = \frac{\lambda}{\mu}$. The average delay per hop is then given by

$$D_{ij} = \frac{2 - p_{ij} \cdot \rho}{2 \cdot \mu \cdot (1 - p_{ij} \cdot \rho)}$$

and the average delay in the network is

$$D = \sum_{i,j} D_{ij} \cdot p_{ij} = \sum_{i,j} \frac{p_{ij} \cdot (2 - p_{ij} \cdot \rho)}{2 \cdot \mu \cdot (1 - p_{ij} \cdot \rho)} \tag{5.6}$$

We provide a comparison between the results obtained by the SR algorithm and the analytical results obtained by the above computation. The purpose of this comparison is to show that the proposed algorithm is robust to the inaccuracy of the performance index evaluation. These results are obtained for the hypercube and the shufflenet topologies.

Hypercube Topology

Although the node degree increases logarithmically with the number of nodes N, due to its simplicity, small diameter, easy routing, and high parallelism, the hypercube is one of the most investigated topologies for multiprocessor systems [46]. We believe that the hypercube also has a big potential as the virtual topology (Host Graph) of WDM based networks.

The traffic pattern (the Image Graph) is given by Figure 5.1: Notice that this demand matrix exactly reflects the hypercube Image Graph with $N = 8$. The average packet arrival rate is $\lambda = 2$ and the packet transmission time is $\frac{1}{\mu} = 5$.

We denote by s_{init} the initial mapping, D_{init} the average delay obtained by the initial mapping, s_{SR}, the mapping obtained by the SR algorithm and the associated delay D_{SR}. The results are shown in Table 5.1.

We observe that the network average delay of a random mapping (that does not account for the traffic patterns) is up to six times higher than the one of the optimal mapping.

The results of these experiments show some interesting properties of our algorithm:

$$P = \begin{pmatrix} 0 & 0.012 & 0 & 0.021 & 0.026 & 0 & 0 & 0 \\ 0.067 & 0 & 0.034 & 0 & 0 & 0.043 & 0 & 0 \\ 0 & 0.059 & & 0.073 & 0 & 0 & 0.067 & 0 \\ 0.033 & 0 & 0.021 & 0 & 0 & 0 & 0 & 0.045 \\ 0.039 & 0 & 0 & 0 & 0 & 0.025 & 0 & 0.076 \\ 0 & 0.048 & 0 & 0 & 0.047 & 0 & 0.056 & 0 \\ 0 & 0 & 0.032 & 0 & 0 & 0.041 & 0 & 0.043 \\ 0 & 0 & 0 & 0.045 & 0.020 & 0 & 0.027 & 0 \end{pmatrix}$$

Figure 5.1: The 8-Node hypercube type traffic pattern

	Initial Random Mapping s_{init}	Aver. Delay D_{init}	Optimal Mapping State s_{SR}	Aver. Delay D_{SR}	Iterations	CPU TIME (in sec.)
1	(1,8,2,6,4,7,5,3)	45.77	(4,8,5,1,3,7,6,2)	7.78	465	64.7
2	(8,6,3,2,5,7,1,4)	32.41	(2,1,5,6,3,4,8,7)	7.24	640	103.2
3	(3,4,8,1,2,6,7,5)	24.27	(3,7,6,2,4,8,5,1)	7.63	478	73.3
4	(2,7,1,8,5,3,6,4)	49.20	(7,3,4,8,6,2,1,5)	7.11	524	88.7

Table 5.1: Results for an 8-Node Hypercube Identical Mapping

1. All the final results in 5.1 can be proved to be optimal by Rule 1. We also notice that due to the symmetrical structure of regular structures, there are multiple optimum states in the states space. For the hypercube, the size of the search space, $|S_N|$, and the size of the optimum subset size, $|O_N|$, are given by:

$$|S_8| = 8! = 40320$$

$$|O_8| = 48$$

The ratio between the optimum number of states and the state space size is given by:

$$\frac{|O_8|}{|S_8|} = 0.001$$

In 100 experiments for the 8 node hypercube, each with a different initial mapping, our algorithm found the optimum states 99 times, and only once it obtained a sub-optimum state.

2. The results of all our experiments indicate that the algorithm we proposed is robust against the error of the system performance estimates. This can be shown by comparing between the optimal state performance index obtained by our short term simulation and the one obtained by analytical model. Substituting $\rho = 10$ and traffic pattern 1 in (6) we obtain the mean packet delay for an 8-Node Hypercube

$$D_{optimum} = 8.095$$

When comparing this result with the simulation results (shown in 5.1), we notice that there is about $5 - 10$ percent difference between them. This is due to the fact that our simulation gathered statistics only from the 200 transmitted packets after discarding the first 100 packets. Although there is some performance difference in our simulation between poor states and good states, the simulated system is far from exhibiting a stationary behaviour. The property of robustness toward accuracy of the performance measure is a key point to the speed superiority of the proposed algorithm.

The curve of system performance versus the number of iterations for an 8-node hypercube identical mapping is shown in Figure 5.1. In the first 100 iterations the system performance is jumping immensely among most of the system states. However, after the system figures out the set of good states, the jumps are reduced, and finally the optimum states are reached.

Shufflenet Topology

We apply our algorithm to an 8-Node shufflenet host graph and a quasi-uniformly distributed shufflenet structure traffic pattern which dictates a shufflenet image graph) is given by Figure 5.2. The system average packet arrival rate is $\lambda = 1$, and the service time in each

$$P = \begin{pmatrix} 0 & 0 & 0 & 0 & 0.062 & 0.035 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.035 & 0.078 \\ 0 & 0 & 0 & 0 & 0.032 & 0.067 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.189 & 0.045 \\ 0.029 & 0.134 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.076 & 0.029 & 0 & 0 & 0 & 0 \\ 0.054 & 0.070 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.032 & 0.033 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 5.2: The 8-Node shufflenet type traffic pattern

	Initial Random Mapping s_{init}	Aver. Delay D_{init}	Optimal Mapping State s_{SR}	Aver. Delay D_{SR}	Iterations	CPU TIME (in sec.)
1	(1,6,3,8,5,7,4,2)	76.23	(2,1,4,3,7,8,5,6)	12.38	1234	128.7
2	(3,2,5,7,6,8,1,4)	64.41	(7,8,5,6,1,2,3,4)	11.24	1465	158.4
3	(3,4,8,1,2,6,7,5)	84.27	(5,6,7,8,1,2,3,4)	7.89	943	95.3

Table 5.2: Results for 8 Node ShuffleNet Identical Mapping

hop is 5.

The results obtained by the SR algorithm are provided in Table 5.2.

The results in 5.2 are also proved to be optimal solutions by Rule 1. The increased number of iterations compared with the previous cases, is explained by the very small ratio between the number of optimum states and the total number of states, which is given by:

$$\frac{|O_8|}{|S_8|} = \frac{4!}{8!} = 0.0006$$

Despite this ratio, the SR algorithm finds the optimal solution in reasonable time. Table 5.2 shows that the mapping optimization process has the same properties as the hypercube identical mapping. Therefore, the same conclusions as in the previous section still apply.

From the system performance point of view, the network average delay has been reduced to as small as one tenth when compared with the random mapping.

Random Traffic Pattern

In this section, the image graph is random, i.e., does not create any regular structure as

$$P = \begin{pmatrix} 0.0000 & 0.0400 & 0.0100 & 0.0300 & 0.0700 & 0.0067 & 0.0067 & 0.0133 \\ 0.0200 & 0.0000 & 0.0100 & 0.0133 & 0.0100 & 0.0500 & 0.0033 & 0.0100 \\ 0.0100 & 0.0500 & 0.0000 & 0.0200 & 0.0067 & 0.0200 & 0.0200 & 0.0133 \\ 0.0100 & 0.0100 & 0.0100 & 0.0000 & 0.0033 & 0.0167 & 0.0167 & 0.0300 \\ 0.0200 & 0.0067 & 0.0233 & 0.0100 & 0.0000 & 0.0200 & 0.0167 & 0.0400 \\ 0.0067 & 0.0100 & 0.0067 & 0.0033 & 0.0200 & 0.0000 & 0.0300 & 0.0000 \\ 0.0100 & 0.0100 & 0.0500 & 0.0033 & 0.0067 & 0.0400 & 0.0000 & 0.0200 \\ 0.0100 & 0.0167 & 0.0100 & 0.0400 & 0.0200 & 0.0100 & 0.0100 & 0.0000 \end{pmatrix}$$

Figure 5.3: A random generated traffic pattern

	Aver. Delay D_{init}	Aver. Delay D_{SR}	Iterations	CPU TIME (in sec.)
1	25.77	14.38	363	44.7
2	30.23	11.70	465	46.2
3	27.74	12.31	391	43.3
4	23.20	12.11	424	38.7

Table 5.3: Results for an 8-Node Random Traffic Pattern Mapping

in the previous subsections. The traffic pattern is generated by a weighted random function which is obtained by the product of a uniform distribution and a weight distributed in the interval [1..3]. The traffic pattern is given by Figure 5.3. The system average packet arrival rate is $\lambda = 2$, and the service time in each hop is 5. The host graph is an 8-Node hypercube. The results are given in Table 5.3.

Despite the fact that we can not prove the outputs of our algorithm are optimum, a significant improvement in the performance index (up to 65%) can be observed in Table 5.3 when compared with the initial random mapping.

5.5 Conclusions

We formulated the wavelength assignment problem into a node mapping optimization problem, and proposed an algorithm to solve this NP-complete problem. The results of our investigation show that the proposed algorithm is robust against the simulation error and as a result it is sufficiently fast for online use. In all cases, our results show that a significant performance improvement is obtained when compared with a random wavelength mapping. Furthermore the analyses show that the output of our algorithm provides the top configurations (up to top 10^{-11}) of all possible configurations.

Bibliography

[1] G. R. Ash, R. H. Cardwell, and R. P. Murray, "Design and Optimization of Networks with Dynamic Routing," *Bell Systems Technical Journal*, Vol. 60, No. 8, 1981, pp. 1787-1820.

[2] G. R. Ash, A. H. Kafker, and K. R. Krishnan, "Servicing and Real-Time Control of Networks with Dynamic Routing," *Bell System Technical Journal*, Vol. 60, No. 8, 1981, pp. 1821-1845.

[3] M.H. DeGroot, *Optimal Statistical Decisions*, New York: McGraw-Hill, 1970.

[4] D. Bertsekas and R. Gallager, *Data Networks,* Englewood Cliffs: Prentice Hall, 1992.

[5] S. G. Eick, "Surveillance Strategies for a Class of Adaptive-Routing Algorithms in Circuit-Switched DNHR Communications Networks: A Simulation Study," *IEEE Journal on Selected Areas in Communications*, Vol. 6, No. 4, 1988, pp. 760-765.

[6] A. Ganz, W. B. Gong, C. M. Krishna, and W. Zhai, "Reconfiguration of Ring Networks Using the Stochastic Ruler Algorithm," *IEEE Conference on Decision and Control*, 1992, pp. 3221-3226.

[7] M. G. Kendall and A. Stuart, *The Advanced Theory of Statistics, Volume 2: Inference and Relationship*, New York: Hafner Publishing, 1966.

[8] S. M. Ross, *Stochastic Processes*, New York: John Wiley and Sons, 1983.

[9] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines.* John Wiley & Sons, Inc., 1989.

[10] P. Bratley, B. L. Fox, and L. E. Schrage. *A Guide to Simulation.* Springer-Verlag, second edition, 1987.

[11] S. H. Brooks. A discussion of random methods for seeking maxima. *Operations Research*, 6, 1958.

[12] K. L. Buescher and P. R. Kumar. Learning by canonical smooth estimation, part I: Simultaneous estimation. To appear in *IEEE Transactions on Automatic Control*,

`http://black.csl.uiuc.edu:80/~prkumar/`

[13] K. L. Buescher and P. R. Kumar. Learning by canonical smooth estimation, part II: Learning and Choice of Model Complexity. To appear in *IEEE Transactions on Automatic Control,*

`http://black.csl.uiuc.edu:80/~prkumar/`

[14] D. P. Connors and P. R. Kumar. Simulated annealing type markov chain and their order balance equations. *SIAM J. Control and Optimization,* 27(6):1440–1461, November 1989.

[15] F. Darema, S. Kirkpatrick and A. Norton. Parallel algorithms for chip placement by simulated annealing. *IBM Journal of Research and Development,* 31, 3, 391-402, May, 1987.

[16] M. Durand and S.R. White. Permissible error in parallel simulated annealing. *Technical Report RC 15487,* IBM T. J. Watson Research Center, Yorktown Heights, New York, 1990.

[17] S.B. Gelfand and S.K. Mitter. Simulated annealing with noisy or imprecise energy measurements. *Journal of Optimization Theory and Applications,* 62, 1, 49–62, July, 1989.

[18] S.B. Gelfand and S.K. Mitter. Simulated annealing type algorithms for multivariate optimization. *Algorithmica,* 6, 3, 419–436, 1991.

[19] D. Goldsman and B. Nelson. Ranking, Selection Method for Stochastic Optimization Problem. In *Proceedings of 1994 Winter Simulation Conference,1994.*

[20] W-B. Gong and Y.C. Ho and W. Zhai. Stochastic Comparison Algorithm for Discrete Optimization with Estimation. In *Proc. of the 31st on CDC,* Tucson, Arizona, Dec., 1992.

[21] D.R. Greening. Simulated Annealing with Inaccurate Cost Functions. *Technical Report,* Department of Computer Science, University of California, Los Angeles.

[22] D.R. Greening and F. Darema. Rectangular spatial decomposition methods for parallel simulated annealing. In *Proceedings of the International Conference on Supercomputing,* Crete, Greece, 295–302, June 1989.

[23] L.K. Grover. Simulated annealing using approximate calculation. In *Progress in Computer Aided VLSI Design,* Vol. 6, Ablex Publishing Corp., 1989.

[24] S.S. Gupta and S. Panchapakesan. *Multiple Decision Procedures: Theory and methodology of selecting and ranking populations.* Wiley, New York, 1979.

[25] B. Hajek. A Tutorial Survey of Theory and Applications of Simulated Annealing. In *Proc. of the 24th CDC*, Ft.Lauderdale, Florida, Dec. 1985.

[26] Y.C. Ho, R. Sreenivas, and P. Vakili. Ordinal Optimization of DEDS. *J. of DEDS*, April 1992.

[27] D. L. Isaacson and R. W. Madsen. *Markov Chains: Theory and Applications.* John Wiley & Sons, Inc., 1976.

[28] R. Jayaraman and F. Darema. Error tolerance in parallel simulated annealing techniques. In *Proceedings of the International Conference on Computer Design*, IEEE Computer Society Press, 545–548, 1988.

[29] T. Lai and S. Yakowitz *Machine Learning and Nonparametric Bandit Theory. IEEE Trans. on Automatic Control*, 40(7):1199–1209, July 1995.

[30] D. Mitra, F. Romeo, and A. Sansiovanni-Vincentelli. Convergence and Finite-Time Behavior of Simulated Annealing. In *Proc. of the 24th CDC*, Ft.Lauderdale, Florida, Dec. 1985.

[31] F. Romeo and A. Sangiovanni-Vincentelli. A theoretical framework for simulated annealing. *Algorithmica*, 6, 3, 302–345, 1991.

[32] T.J. Santner and A.C. Tamhane *Design of Experiments: Ranking and Selection* Marcel Dekker, 1984

[33] S. Yakowitz and T. Jayawardena and S. Li *Theory for Automatic Learning under Partially Observed Markov-dependent Noise. IEEE Trans. on Automatic Control*, 37(9):1316–1324, September 1992.

[34] D. Yan and H. Mukai. Stochastic discrete optimization. *SIAM J. on Control and Optimization*, 30(3):549–612, May 1992.

[35] C. A. Brackett, "Dense Wavelength Division Multiplexing Networks: Principles and Applications" *IEEE Journal on Selected Areas in Communications, Vol. 8, No.6, August 1990.*

[36] A. Ganz and Z. Koren, "WDM Passive Star-Protocols and Performance Analysis", *IEEE Infocom'91*, Florida, April 1991.

[37] M. S. Goodman, H. Kobrinski, M. P. Vecchi, R. M. Bulley and J. L. Gimlett, "The LAMBDANET Multiwavelength Network : Architecture, Applications, and Demonstrations", *IEEE Journal on Selected Areas in Communications*, Vol. 8, No. 6, August 1990.

[38] A. S. Acampora, "A Multichannel Multihop local Lightwave Network" *IEEE GLOBE-COM'87, May 1987.*

[39] A. S. Acampora M. J. Karol and M. G. Hluchyj, "Terabit Lightwave Networks: The Multihop Approach" *At&T Technical Journal, Volume 66, Issue 6 ,November/December 1987.*

[40] B. Li and A. Ganz, "Virtual Topologies for WDM Star LANs - The Regular Structures Approach" *IEEE INFOCOM'92.*

[41] J. P. Labourdette and A. S. Acampora, "Logically Rearrangeable Multihop Lightwave Networks" *IEEE Transactions On Communications, Vol. 39, No.8. August 1991.*

[42] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, "Optimization by Simulated Annealing" *Science Vol. 220, No. 4598, May 1983.*

[43] D. Yan and H. Mukai, "Discrete Optimization With Estimation" *Proceedings of the 28th Conference on Decision and Control, Tampa ,Florida, December 1989.*

[44] Y. C. Ho, R. S. Sreenivas, P. Vakili "Ordinal Optimization of DEDS" *to appear in Journal of Discrete Event Dynamic System*, 1992.

[45] H. A. David "Order Statistic" *John Wiley & Sons, Inc.*, 1981.

[46] L. N. Bhuyan and D. P. Agrawal "A general class of processor interconnection strategies" *in proc. 9th Annu Int. Symp on Compt. Arch., Austin, TX* April 1982.

[47] M. Chean and J. A. B. Fortes, "A Taxonomy of Reconfigurable Techniques for Fault-Tolerant Processor Arrays", *IEEE Trans. Computers*, Vol. 39, No. 1, 1990.

[48] M. Choi and C. M. Krishna "An Adaptive Algorithm to Ensure Differential Service in a Token-Ring Network", *IEEE Trans. Computers,* Vol. 39, January 1990, pp. 19-33.

[49] B. Hajek, "A Tutorial Survey of Theory and Applications of Simulated Annealing," *Proceedings of the 24th CDC*, pp. 755-760, 1985.

[50] Y. C. Ho, R.S.Sreenivas and P. Vakili, "Ordinal Optimization of DEDS", *J. of DEDS*, April, 1992.

[51] S. H. Hosseini, "On Fault-Tolerant Structure, Distributed Fault-diagnosis, Reconfiguration and Recovery of the Array Processors", *IEEE Trans. Computers*, Vol. 38, No. 7, 1989.

[52] T. Ibaraki and N. Katoh, "Resource Allocation Problems", The MIT Press, 1988.

[53] "IEEE Standards for Local-Area Networks: Token-Ring Access Method and Physical Layer Specifications", ANSI/IEEE Standard 802.5-1985 (ISO DP 8802/5, ECMA 89, 1983), 1985.

[54] M. B. Ignatev, T. M. Denisova, M. S. Kupriyan, O. N. Yargin, "Adaptive Reconfiguring Algorithms for Systems with Redundancy", *Cybernetics*, Vol. 24, No. 5, 1988.

[55] H. Li and Q. F. Stout, "Reconfigurable SIMD Massively Parallel Computers", *Proc. IEEE*, Vol. 79, No. 4, 1991.

[56] M. B. Lowrie and W. K. Fuchs, "Reconfigurable Tree Architectures Using Subtree-Oriented Fault-Tolerance", *IEEE Trans. Computers*, Vol. 36, No. 10, 1987.

[57] M. Malek and E. Opper, "The Cylindrical Banyan Multicomputer – A Reconfigurable Systolic Architecture", *Parallel Computing*, Vol. 10, No. 3, 1989.

[58] P. S. Matharu and S. S. Lawson, "Reconfigurable Programmable Architecture for Cascaded Unit Elements and Lattice WDF's", *IEE Proc., Ser. G*, Vol. 135, No. 5, 1988.

[59] R. Miller, V. K. Prasanna Kumar, D. Reisis and Q. F. Stout, "Parallel Computations on Reconfigurable Meshes", *IEEE Trans. Computers*, Vol. 39, 1990.

[60] D. K. Pradhan, "Dynamically Restructurable Fault-Tolerant Processor Network Architectures", *IEEE Trans. Computers*, Vol. C-34, No. 5, 1985.

[61] D. Sitaram, I. Koren and C. M. Krishna, "A Random Distributed Algorithm to Embed Trees in Partially Faulty Processor Arrays", *J. Parallel and Distributed Computing*, Vol. 12, No. 1, 1991.

[62] D. Yan and H. Mukai, "Discrete Optimization with Estimation", *Proceedings of the 28th CDC*, Tampa, Florida, pp.2463-2468, 1989.